UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOGADORES VIRTUAIS PARA O JOGO WAR

Inteligência Artificial e Jogos

por

Rodrigo Bona

Rudimar Luís Scaranto Dazzi, M.Sc.
Orientador

Itajaí (SC), Junho de 2007

UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOGADORES VIRTUAIS PARA O JOGO WAR

Inteligência Artificial e Jogos

por

Rodrigo Bona

Relatório apresentado à Banca Examinadora do Trabalho de Conclusão do Curso de Ciência da Computação para análise e aprovação.
Orientador: Rudimar Luís Scaranto Dazzi, M.Sc.

Itajaí (SC), Junho de 2007

DEDICATÓRIA

Dedico o meu trabalho, primeiramente a Deus, a minha família, incluindo os meus amigos e especialmente minha esposa, pois sem eles não teria forças para realizá-lo.

AGRADECIMENTOS

Agradeço a Deus por colocar as pessoas certas na hora que mais precisei, e por me mandar o seu espírito santo pra iluminar o meu caminho.

Agradeço também aos meus pais, Altair e Sonia, pois é por causa dos esforços deles que pude estudar informática e seguir nessa área, como também por toda minha formação pessoal que formou a pessoa que sou hoje.

Agradeço especialmente a minha esposa Márcia, que sempre teve fé em mim e na minha capacidade, que esteve ao meu lado, principalmente nos momentos difíceis, que me deu forças para que eu pudesse levantar e seguir em frente.

Agradeço ainda ao meu amigo Marcos, que foi uma pessoa que conheci por acaso na faculdade, e se tornou uma das únicas amizades verdadeiras que tenho na vida, não esquecendo de sua noiva Zahie, que entrou na família a pouco tempo, mas que é considerada com uma velha amiga.

Não posso esquecer todos os professores que, de uma forma ou de outra, agregaram conhecimento e lições que levarei para o resto de minha vida. Dentre eles destaco o professor Rudimar, que além de me orientar nesse trabalho, e me incentivar quando pensava em desistir, mostrou-se um amigo em que podemos confiar. Agradeço a professora Anita, que com o seu jeito severo, me mostrou que mesmo com as adversidades devemos continuar em frente e não desistir.

E por fim agradeço a todos que, as vezes, com apenas uma palavra me apoiaram e depositaram fé no meu trabalho e nas minhas ações.

Obrigado.

SUMÁRIO

LISTA DE ABREVIATURAS vi
LISTA DE FIGURAS vii
LISTA DE TABELAS viii
RESUMO ix
ABSTRACT x
1 INTRODUÇÃO 1
1.1 PROBLEMATIZAÇÃO 3
1.1.1 Formulação do Problema 3
1.1.2 Solução Proposta 3
1.2 OBJETIVOS 3
1.2.1 Objetivo Geral 3
1.2.2 Objetivos Específicos 4
1.3 METODOLOGIA 4
1.4 ESTRUTURA DO TRABALHO 5
2 FUNDAMENTAÇÃO TEÓRICA 7
2.1 O JOGO WAR 7
2.1.1 Histórico 7
2.1.2 Componentes do Jogo 7
2.1.3 Funcionamento do Jogo 8
2.1.4 Regras 9
2.2 CONEXÃO MULTIPLAYER 11
2.2.1 Modos Cliente e Servidor 11
2.3 OUTROS JOGOS 14
2.4 INTELIGÊNCIA ARTIFICIAL 15
2.4.1 Sistemas Especialistas 18
2.4.2 Ferramentas de Inferência 21
2.4.3 Agentes Inteligentes 21
2.4.4 Comunicação entre Agentes 28
2.4.5 FIPA 29
2.4.6 JADE 30
2.4.7 Linguagem JAVA 36
3 DESENVOLVIMENTO 38
3.1 MODELAGEM 38
3.1.1 Requisitos do Sistema 38
3.1.2 Casos de Uso 42
3.1.3 Árvore de Decisão 45
3.1.4 Análise de Riscos 48

3.2 IMPLEMENTAÇÃO 49
3.2.1 Implementação dos Agentes 49
3.2.2 Implementação das Regras 52
3.2.3 Usando o JCLIPS 52
3.2.4 Comunicação entre Sistemas 55
3.2.5 Servidor 55
3.2.6 Cliente 58
4 CONCLUSÃO 63
4.1 TRABALHOS FUTUROS 66
REFERÊNCIAS BIBLIOGRÁFICAS 67
5 GLOSÁRIO 72
Diagramas 74
A.1 DIAGRAMAS DE ATIVIDADES 74
A.1.1 Inicializa Jogo (Servidor) 74
A.1.2 Inicializa Jogo (Cliente) 75
A.1.3 Partida 75
A.2 DIAGRAMAS DE ESTADO 77
A.2.1 Iniciar Jogo 77
A.2.2 Durante o Jogo 78
A.2.3 Comunicação entre Computadores 79
A.3 DIAGRAMA DE SEQÜÊNCIA 80
A.4 ÁRVORES DE DECISÃO 81
A.4.1 Árvore Genérica 81

LISTA DE ABREVIATURAS

ACC	Agent Communication Channel
AMS	Agent Management System
CLIPS	C Language Integrated Production System
DF	Directory Facilitator
FIPA	Foundation for Intelligent Physical Agents
GUI	Graphical User Interface
GUID	Globally Unique Identifier
IA	Inteligência Artificial
IP	Internet Protocol
JADE	Java Agent Development Framework
JRE	Java Run-time Enviroment
JVM	Java Virtual Machine
NASA	National Aeronautics and Space Admistration
QI	Quoeficiente de Inteligência
RBC	Raciocínio Baseado em Casos
RMI	Remote Method Invocation (Invocação Remota de Métodos)
SBC	Sistemas Baseados em Conhecimento
SE	Sistema Especialista
TCC	Trabalho de Conclusão de Curso
UNIVALI	Universidade do Vale do Itajaí
USP	Universidade de São Paulo
HTN	Hierarchical Task Networks

LISTA DE FIGURAS

Figura 1. Mapa do Jogo WAR 7
Figura 2. Outros Componentes do Jogo WAR: (a) exércitos; (b) cartas território; (c) cartas objetivo. 8
Figura 3. Escopo da conexão *multiplayer* 11
Figura 4. Cliente Socket Java usando a classe TCPClient 13
Figura 5. Servidor Socket Java utilizando a classe TCPServer 13
Figura 6. Arquitetura básica dos sistemas especialistas. 20
Figura 7. Características de Agentes 24
Figura 11. Diagrama de *USE CASE* 43
Figura 12. Diagrama de Classes 45
Figura 13. Árvore de decisão para atacar um outro exército 46
Figura 14. Árvore de Decisão – Atacar Outro Exército – Reposição 47
Figura 15. Árvore de Decisão – Atacar Outro Exército – Atacar 47
Figura 16. Árvore de Decisão – Atacar Outro Exército – Movimentar 48
Figura 17. Código para o comportamento padrão do agente 50
Figura 18. Código para criar e iniciar um agente JADE 51
Figura 19. Classes para instanciar o agente no jogo 51
Figura 20. Código de Exemplo de Conexão com o JCLIPS 52
Figura 21. Regra em JCLIPS 53
Figura 22. Regras para calcular de média 53
Figura 23. Interface e Classe para árvore de decisão 54
Figura 24. Tela principal do Servidor 57
Figura 25. Tela Cria Servidor 57
Figura 26. Aguardando conexão de outros jogadores 58
Figura 27. Tela principal do programa cliente 60
Figura 28. Conectar com o Servidor 60
Figura 29. Mapa do Jogo 61
Figura 30. Diagrama de Atividade - Inicializa Jogo (Servidor) 74
Figura 31. Diagrama de Atividade - Inicializa Jogo (Cliente) 75
Figura 32. Diagrama de Atividade – Partida 76
Figura 33. Diagrama de Estado - Iniciar Jogo 77
Figura 34. Diagrama de Estado - Durante o Jogo 78
Figura 35. Diagrama de Estado - Comunicação Entre Computadores 79
Figura 36. Diagrama de Seqüência – Comunicação Cliente-Servidor 80
Figura 37. Árvore de Decisão – Árvore Genérica 81

LISTA DE TABELAS

Tabela 1. Trocas de cartas por exércitos.....10
Tabela 2. Características de Outros Jogos X WAR.....15
Tabela 3. Agente Inteligente X Jogadores Virtuais.....27

RESUMO

BONA, Rodrigo. **Jogadores Virtuais para o Jogo War.** Itajaí, 2007. 92 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação)–Centro de Ciências Tecnológicas da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí, 2007.

WAR é um jogo do gênero de estratégia, onde em um tabuleiro com o mapa mundi impresso, os jogadores disputam territórios com seus exércitos, a fim de cumprir um determinado objetivo e, dessa forma, vencer a partida. O jogo WAR apresenta diversas regras, que devem ser cumpridas no decorrer da partida, sendo elas um obstáculo para jogadores iniciantes. Outra dificuldade encontrada nesse jogo é que, devido ao seu gênero e aliado com a quantidade mínima de jogadores, exige dos participantes um certo nivelamento em suas habilidades estratégicas. O presente projeto criou uma versão eletrônica do jogo WAR, e utilizou para isso a linguagem Java, permitindo que os jogadores possam participar de partidas desse jogo em computadores distribuídos em uma rede, com o emprego da tecnologia de RMI (*Remote Method Invocation*). Além disso, são implementados jogadores virtuais, aos quais poderão ser adicionados na partida para completar a quantidade mínima de jogadores ou mesmo para aumentar a quantidade de participantes. Para isso são aplicadas técnicas de Inteligência Artificial (IA) e Agentes Inteligentes, a fim de fazer os jogadores virtuais representarem jogadores humanos em suas ações, sendo que para representar os conhecimentos desses jogadores foi utilizada a técnica de Sistemas Especialistas (SE), simulando as decisões que uma pessoa tomaria conforme as ações a serem realizadas no jogo.

Palavras-chave: Inteligência Artificial. Agentes Inteligentes. Jogos.

ABSTRACT

WAR is a game of strategy sort, where in a tray with a map printed, the players dispute territories with theirs armies, in order to fulfill an objective and, in this way, win the game. Game WAR presents diverse rules that must be fulfilled along the game, and this represent and obstacle for beginners. Other difficulties that may be found in this game, is the minimum amount of players necessary, therefore the participants must have certain level in its strategical abilities. The present project created an electronic version of game WAR, and used for this the Java language, allowing that players can participate of games in computers distributed in a net, with the employ of the RMI technology (Remote Method Invocation). Moreover, virtual players are implemented, to which can be added in the game to complete the minimum amount of players or even to increase the amount of participants. Therefore, Artificial Intelligence (AI) techniques and Intelligent Agents are applied, in order to make the virtual players to represent human players in its actions, once that to represent the knowledge of these players, the technique of Expert Systems (ES) was applied, simulating the decisions that a person would take according to the actions to be carried through in the game.

Keywords: Artificial Intelligence, Intelligent Agents, Games

1 INTRODUÇÃO

A maioria dos jogos tem por finalidade entreter pessoas, sendo eles eletrônicos ou não. Em grande parte desses jogos, há a possibilidade de se jogar com dois jogadores e, em alguns casos, é difícil achar outra pessoa para poder dar início a uma partida, e ainda mais difícil achar uma pessoa experiente no jogo, ao qual não seja necessário explicar todas as regras desse mesmo jogo e a forma de jogar. Essa dificuldade aumenta quando um jogo tem a quantidade de pessoas mínimas, para dar início a uma partida, como é o caso do jogo WAR, que precisa três ou mais jogadores.

Segundo o manual do usuário do jogo Grow (1997), o jogo trata de conquistas de territórios, por intermédio de batalhas entre soldados, com o intuito de alcançar um objetivo previamente selecionado por cartas. Esse é um jogo de estratégia, onde é praticado em um tabuleiro que tem impresso os continentes, e em cada continente as divisões regionais, que não necessariamente representa a realidade. Para poder dar início a esse jogo há a necessidade de no mínimo três jogadores, chegando ao máximo de seis.

Por ser um jogo que exige tempo e também certo nivelamento estratégico entre os participantes, para que haja uma partida no mínimo interessante, fica difícil arranjar o mínimo de jogadores para realizá-la. Então alguns jogos fazem uso de jogadores virtuais para amenizar esse problema. Um exemplo de jogo que pode fazer uso de jogadores virtuais para poder ser realizada uma partida é o jogo Couter Strike. Esse jogo é em primeira pessoa, ou seja, o jogador não vê a própria personagem, somente a arma e os outros jogadores. Os jogadores são divididos em dois times, um policial e outro terrorista. O objetivo é, se estiver no time policial, impedir que os terroristas plantem uma bomba em certo alvo ou salvar reféns que estão em poder dos terroristas, e caso esteja no time terrorista, o contrário, plantar uma bomba ou não deixar os reféns serem resgatados. Para a realização de uma partida, faz se necessário no mínimo dois jogadores, um em cada time, sendo que quanto mais jogadores participarem mais empolgante fica a partida. Logo, os jogadores virtuais auxiliam nesse exemplo, sendo adicionados ao jogo e permitindo que haja um misto de jogadores virtuais com jogadores humanos.

Esses jogadores virtuais são programas desenvolvidos para interagirem com o ambiente do jogo, e reagem ao que acontece nele. Conforme o exemplo do Counter Strike, os jogadores virtuais, utilizando como exemplo o time policial, sairão atrás dos terroristas para impedir que eles plantem uma bomba ou irão ao resgate de reféns, e reagem conforme as situações que aparecem no decorrer

da partida. Então se aparecer algum terrorista em seu caminho ele tentará eliminá-lo, sendo que para isso ele pode utilizar um ataque direto, esperar que o inimigo saia de onde está ou até escolher outro caminho para passar. Ainda esses jogadores virtuais podem ter níveis de dificuldade, passando do mais fácil, onde se parecem com jogadores iniciantes, ou mais difícil onde parecem ter mais experiência no jogo, e dessa forma permite que vários tipos de jogadores humanos possam jogar, desde os iniciantes até os mais experientes.

Então, este projeto propôs implementar uma versão eletrônica do jogo WAR, com uma interface primária, onde será possível visualizar o mapa do mundo conforme o jogo, dados sobre quantidade de exércitos de cada cor nas regiões, as cartas atuais do jogador e informações de quem ganhou ou perdeu a disputa, e utilizar jogadores virtuais, afim de que uma pessoa possa utilizá-lo sem a necessidade de mais jogadores humanos, ou seja, somente um jogador humano com vários jogadores virtuais, ou mesmo aumentar a quantidade de jogadores, com vários jogadores humanos e vários jogadores virtuais, sendo que os demais jogadores humanos estarão acessando o jogo a partir de outros computadores em um ambiente *multiplayer*.

Para implementar os jogadores virtuais foi utilizado os conceitos de agentes, que conforme Wooldridge (1999), um agente é um software autônomo capaz de executar ações em um ambiente, a fim de alcançar um objetivo. Para classificar um agente deve-se perceber as seguintes características: Reativo, onde o mesmo percebe o ambiente e reage a ele; Pró-Ativo, onde deve atuar direcionado ao objetivo; Social, deve interagir com outros agentes ou com humanos. Sendo assim o jogador virtual é classificado como um agente inteligente, por executar ações no ambiente do jogo, atacar, repor exércitos com o propósito de alcançar um objetivo, e “exibem, de alguma forma, características associadas ao que chamamos inteligência”. (RICH & KNIGHT, 1994)

Foram avaliados jogos que fazem uso de jogadores virtuais, a fim de analisar o funcionamento, técnicas de inteligência artificial empregadas e o método de criação desses jogadores virtuais no jogo.

A execução deste projeto de pesquisa também se justifica em nível de Trabalho de Conclusão de Curso para o Curso de Ciências da Computação, uma vez que trata do desenvolvimento de uma solução computacional que faz uso de muitas tecnologias, conceitos e teorias relevantes a essa área, tais como: utilização de técnicas de inteligência artificial para os jogadores virtuais reagirem as jogadas, tomar decisões e também aprender com jogadas de

adversários; armazenamento dos dados obtidos no jogo (jogadas, movimentos); modelagem de sistemas.

O jogo foi desenvolvido para a plataforma Windows, sendo limitado em até seis jogadores (humanos ou virtuais), e o mínimo de três jogadores (um humano e os demais podem ser humanos e/ou virtuais), podendo ser realizado em apenas uma máquina com somente um jogador humano e os demais virtuais, como também em uma rede local ou internet com vários jogadores (sejam virtuais ou humanos) respeitando o limite mínimo e máximo de jogadores. Tendo dois modos, servidor e cliente, sendo que o modo servidor serve para criar um novo jogo e definir a quantidade de jogadores humanos e virtuais que será utilizado no jogo. O modo cliente serve para conectar um outro jogador humano, que estará em outro computador ligado em rede, a um jogo criado pelo modo servidor, a fim de possibilitar a participação de outros jogadores.

1.1 PROBLEMATIZAÇÃO

1.1.1 Formulação do Problema

Para a execução de uma partida no jogo WAR, faz-se necessário a participação de 3 ou mais jogadores, sendo um máximo de 6, onde para ter uma partida no mínimo interessante, os participantes devem ter o mesmo nível de conhecimento no jogo. Então para iniciar uma partida são necessários 3 jogadores ou mais e um balanceamento no nível dos mesmos.

1.1.2 Solução Proposta

Com o intuito de facilitar o início de uma partida, o presente projeto propôs o desenvolvimento de uma versão eletrônica do jogo WAR com jogadores virtuais, que participam de uma partida preenchendo assim o número mínimo de jogadores, sendo que esses jogadores tem níveis de dificuldade afim de nivelar-se com os demais jogadores humanos.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O objetivo geral deste projeto é desenvolver uma versão eletrônica do jogo WAR, implementando jogadores virtuais, para vários jogadores simultâneos.

1.2.2 Objetivos Específicos

Para atingir o objetivo geral proposto, foram alcançados os seguintes objetivos específicos:

- Analisar e implementar as regras do jogo;
- Analisar jogos que fazem uso de jogadores virtuais;
- Pesquisar técnicas para criação de diversos jogadores virtuais em uma mesma partida, definindo qual será utilizada;
- Pesquisar e selecionar técnicas para conexão externa de outros computadores;
- Definir as técnicas de IA para a implementação dos jogadores virtuais;
- Modelagem do processo decisório dos jogadores virtuais;
- Modelagem do Jogo;
- Implementar o Jogo;
- Testar e validar o jogo.

1.3 Metodologia

Os jogos que fazem uso de jogadores virtuais foram analisados quanto a forma que esses jogadores são incorporados aos jogos, se são externos a eles ou já estão implícitos no jogo, e como se dá a inclusão dos mesmos em uma partida, se durante a partida pode se incluso mais jogadores ou se somente no início de cada partida.

As técnicas para criação de diversos jogadores virtuais levaram em conta que no uso de orientação a objetos é necessário apenas criar uma classe jogador virtual, e no jogo basta instanciar essa classe quantas vezes forem necessárias.

Quanto às técnicas para conexão entre computadores, foi levado em conta a forma que os jogos se conectam, por meio de endereços *Internet Protocol* (IP) e *sockets.*

As técnicas de Inteligência Artificial (IA) para implementação dos jogadores virtuais foram definidas levando em conta que os jogadores virtuais possuem as características de agentes, e que a forma como eles interagem no jogo é definido pelo objetivo escolhido.

A modelagem do processo decisório foi realizada conforme cada objetivo existente no jogo, e o que pode ser feito para alcançar esse objetivo.

A modelagem do jogo foi desenvolvida no modelo de UML, usando os diagramas de caso de uso, de classes, de seqüência.

A implementação do jogo foi utilizado a interface de desenvolvimento NetBeans 5.0 da Sun Microsystens, e a linguagem de programação JAVA, onde foi utilizado RMI para permitir a comunicação entre cliente e servidor, o *framework* JADE para implementar os jogadores virtuais e também a API JCLIPS para manter as regras que formam a inteligência.

Os testes foram realizados conforme o desenvolvimento de novas funcionalidades. A execução foi realizada em um computador, executando um servidor e dois a três clientes. Após o sistema permitir a execução de uma partida completa, foram utilizados três computadores, sendo que um deles iniciou o servidor e um cliente, e os demais somente o sistema cliente.

1.4 Estrutura do trabalho

O presente documento está dividido em quatro capítulos, sendo que no capítulo 1, está a Introdução, que traz uma abordagem geral do trabalho, o problema relatado, os objetivos a serem alcançados e a metodologia empregada.

O Capítulo 2, Fundamentação Teórica, apresenta uma revisão bibliográfica sobre as regras do jogo WAR, forma de conexão entre computadores voltada ao jogo, Inteligência Artificial, o qual será empregado a técnica de Sistemas Especialistas para modelar o conhecimento dos jogadores virtuais, Agentes Inteligentes, a forma de comunicação entre eles, os padrões definido pela FIPA para o desenvolvimento de agentes inteligentes e o *framework* JADE para auxiliar na implementação dos agentes inteligentes.

O Capítulo 3, Desenvolvimento, apresenta a metodologia para desenvolver o jogo, incluindo requisitos funcionais e não funcionais, diagramas de *USE CASE* e CLASSE, juntamente com as árvores de decisão utilizadas na base de conhecimento do jogador virtual, análise de riscos, implementação onde é utilizado a linguagem JAVA, comunicação entre os sistemas, como também o funcionamento dos sistemas cliente e servidor.

O Capítulo 4, Conclusão, é feito um fechamento do trabalho, onde são avaliados as técnicas utilizadas como também as dificuldades encontradas e suas resoluções. O texto ainda inclui nove apêndices que completam as informações apresentadas.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 O Jogo WAR

O jogo WAR é classificado como um jogo de estratégia, ou seja, os jogadores participantes devem utilizar os meios disponíveis para alcançar o seu objetivo. (GROW, 1997).

2.1.1 Histórico

Ao final dos anos 60, quatro alunos da faculdade de engenharia da Poli - Universidade de São Paulo (USP) após a formatura decidem iniciar um negócio próprio. Após observarem o mercado, constatam que o mercado para jogos adultos era promissor. Então em agosto de 1972 criam a empresa GROW Jogos e Brinquedos S.A., sendo o seu primeiro produto o jogo WAR e tornou-se o primeiro jogo do Brasil destinado para o público adulto.

2.1.2 Componentes do Jogo

Inicialmente há um mapa (Figura 1) semelhante ao mapa do mundo, dividido em seis continentes sendo que cada continente também é dividido em um determinado número de territórios.

Figura 1. Mapa do Jogo WAR

Fonte: Grow (1997)

Juntamente com o mapa há outros elementos (Figura 2) que compõe o jogo sendo eles:

- Exércitos: são peças coloridas que representam onde o jogador está situado no jogo e o seu poder de ataque e defesa;
- Cartas território: onde cada território do mapa contém a sua carta correspondente;
- Cartas objetivo: onde estão impressos os objetivos ao qual cada jogador terá que alcançar conforme a carta que escolher.

(a)

(b)

(c)

Figura 2. Outros Componentes do Jogo WAR: (a) exércitos; (b) cartas território; (c) cartas objetivo.
Fonte: Grow (1997)

2.1.3 Funcionamento do Jogo

Segundo a Grow (1997), cada participante deve escolher um exército da cor que preferir. Logo após é sorteado uma carta objetivo para cada participante, que por sua vez não poderá revelá-lo aos demais jogadores até o final da partida. Em seguida cada participante joga um dado, sendo que aquele que obtiver o maior valor deverá distribuir as cartas território. Esse processo deverá iniciar-se por si mesmo e em seguida ao jogador mais a esquerda. Após a distribuição das cartas território, cada participante deve colocar um exército no território correspondente a carta que recebeu, sendo que ao final desse processo todos os territórios devem estar ocupados com o exército de um dos participantes e todas as cartas território devem ser agrupadas todas em mesmo monte, onde devem permanecer por todo o jogo.

A rodada inicia com o jogador seguinte ao que recebeu a última carta território. Na primeira rodada cada jogador deve receber e colocar exércitos nos seus devidos territórios de acordo com a sua estratégia e cada um na sua vez. A partir da segunda rodada, o jogo segue uma regra básica, o

jogador recebe exércitos e os distribui nos territórios de seu poder, ataca os adversários se desejar, desloca os exércitos conforme conveniência e recebe carta se fizer jus a ela.

2.1.4 Regras

Conforme Grow (1997), o jogo segue uma série de regras quanto a recebimento de exércitos, ataque á oponentes, movimentação, conquista e troca de cartas e eliminação de um concorrente.

O recebimento de exércitos segue a seguinte regra: ao início de cada jogada, o jogador deve contar a quantidade de territórios que estão em seu poder, devendo dividir esse valor por dois e a quantidade inteira resultante será a quantidade de exércitos que esse jogador pode distribuir em qualquer território de seu poder, nesse momento o jogador deve verificar se há a possibilidade de fazer trocas de cartas, onde poderá obter mais exércitos e distribuí-los nos territórios.

Após distribuir os exércitos nos territórios, o jogador deve escolher onde deseja jogar. As jogadas se restringem em ataque e defesa. O jogador anuncia onde deseja atacar e com quantos exércitos pretende fazê-lo. O ataque segue as seguintes regras:

- O território que está atacando deve ter no mínimo dois exércitos, o de ocupação e o de ataque. O exército de ocupação serve para defender o território e identificar que o ocupa. O exército de ataque para, em caso de vitória, ocupar o território atacado;
- O atacante pode realizar o ataque com quantos exércitos quiser, sendo que deve fazê-lo sempre usando o máximo de três por cada vez;
- O ataque se dará com o sorteio de números entre 1 e 6, onde a quantidade de números a serem sorteados representam a quantidade de exércitos do jogador até 3 exércitos para os dois lados, ataque e defesa. A análise da jogada é realizada com o comparativo dos números sorteados entre si, sendo feita em ordem crescente, ou seja, o maior resultado dos números sorteados para o atacante com o maior resultado dos números sorteados para o defensor, e assim sucessivamente;
- O atacante só pode atacar territórios que fazem divisa com o território de ocupação do mesmo, ou que estejam ligados por um pontilhado, e com a quantidade de exércitos que tiver nesse território;

- O atacante pode realizar quantas batalhas quiser, até que haja somente um exército em cada território ou até o jogador parar a sua jogada; e
- Após cada ataque, o atacante pode invadir o território conquistado apenas com os exércitos que participaram da batalha e que sobreviveram a ela.

Ao término da jogada o jogador tem a possibilidade de movimentar seus exércitos se assim desejar. Para isso o jogador pode transferir um exército de um território á outro somente se ele fizer divisa ou for ligado por pontilhados, sendo que cada exército não pode ser movimentado mais do que um território por rodada.

Ao final de sua jogada, caso o jogador tenha conquistado algum território, é concedida a primeira carta território do monte onde deve ser guarda até uma eventual troca. Cada carta território traz consigo um símbolo que pode ser um quadrado, círculo ou triângulo. A combinação desses símbolos, ou seja, três cartas cada uma com um símbolo ou todas as três com o mesmo símbolo, da ao jogador a possibilidade de receber mais exércitos. No meio das cartas território existe duas cartas coringa, que contem todos os três símbolos impressos, que pode ser combinado para fazer uma troca. Essa combinação só pode ser efetuada no inicio da jogada, antes dos ataques, onde o jogador deve devolver as cartas que estiver utilizando no processo. A quantidade de exércitos que o jogador vai receber depende de quantas trocas foram realizadas no jogo independente de qual jogador tenha efetuado a troca (Tabela 1). Se dentre as cartas território usadas na troca, houver uma que, cujo o território representado está em seu poder, o jogador ganha dois exércitos que devem ser colocados naquele território. O jogador pode acumular até cinco cartas território, após isso é imperativo que o jogador faça uma troca.

Tabela 1. Trocas de cartas por exércitos

Número da Troca	**Quantidade de Exércitos**	**Número da Troca**	**Quantidade de Exércitos**
1º	4	5º	12
2º	6	6º	15
3º	8	7º	20
4º	10	Em seguida 25,30,35,40,45.....exércitos	

Fonte: Grow (1997)

Em caso de um jogador eliminar o outro, as cartas que estiverem de posse do perdedor passam para o ganhador, no caso de o total das cartas território serem maior que cinco, devem ser retiradas às cartas em excesso, sendo que essa retirada deve ser feita pelo jogador seguinte.

Também faz parte do jogo o sigilo, onde é imprescindível que algumas informações sejam ocultas dos outros jogadores, como o seu objetivo como também as cartas que foram recebidas para troca. Uma forma de fazer com que seja garantido esse sigilo é fazer com que cada jogador participe do jogo em computadores diferentes. Para isso o jogo deve permitir que vários jogadores se conectem a mesma partida a partir de uma conexão multiplayer, como é apresentado a seguir.

2.2 Conexão Multiplayer

O jogo permite que até seis jogadores humanos participem da mesma partida. Onde cada jogador deve estar em seu próprio computador, que deverá estar em rede com os demais participantes e o servidor.

2.2.1 Modos Cliente e Servidor

O jogo foi desenvolvido com dois módulos, um cliente e outro servidor (Figura 3). O módulo servidor é utilizado para criar uma nova partida. Neste módulo são informados quantos jogadores participarão do jogo, quais serão jogadores virtuais e quais humanos. Se houver mais de um jogador humano, o servidor fica em modo de espera, até que todos os jogadores humanos estejam conectados ao servidor, e permitindo assim o início da partida.

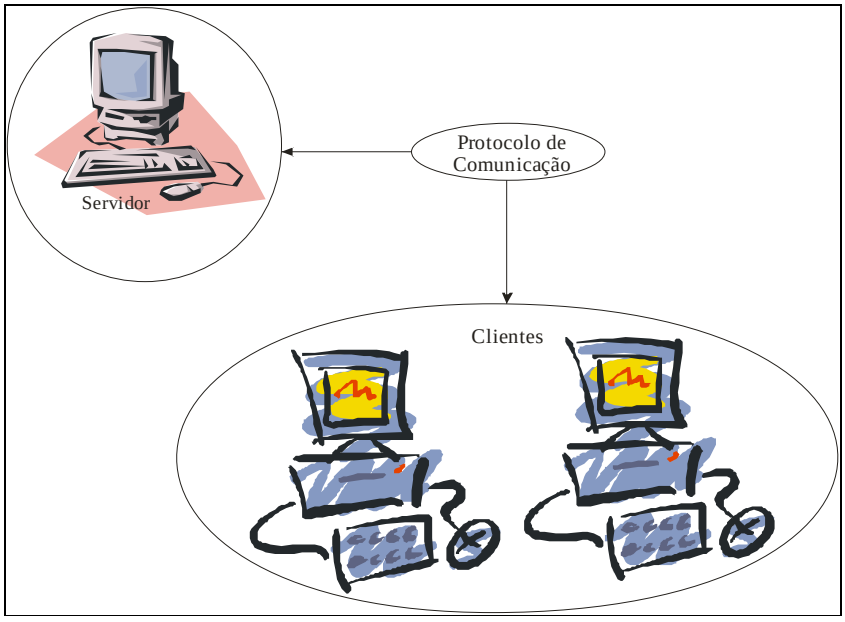

Figura 3. Escopo da conexão *multiplayer*

O módulo cliente serve para conexão com o servidor, sendo informado o IP (*Internet Protocol*) da máquina servidora, permitindo que o cliente possa se conectar com o servidor. O cliente envia mensagens para o servidor informando que deseja fazer uma conexão, e o servidor retorna informando ao cliente se a conexão pode ser estabelecida e se as características do jogador foram aceitas.

Em ambos os módulos o jogador define o nome, ao qual representará o mesmo no jogo e qual a cor de exército que deseja utilizar.

A conexão entre cliente e servidor se dá com a utilização de *sockets.* Conforme Franco & Yoshida (2004), *sockets* são portas lógicas onde programas distribuídos em uma rede podem se comunicar. Esses *sockets* são numerados, variando de 0 à 65536, para utilizar uma dessas portas lógicas basta reservar um número. Para alguns desses *sockets* são reservados serviços como acesso a web, envios e recebimentos de e-mails, entre outros. Por exemplo, para envio de e-mail o programa gerenciador de e-mails deve usar a porta lógica 25, quanto para receber e-mails o mesmo programa deve utilizar a porta lógica 110, já o navegador de Internet se utiliza da porta 80.

Segundo Franco & Yoshida (2004), para a utilização de *sockets* basta haver processo, que deve estar rodando e esperando solicitações na porta lógica que está ocupando, sendo esse chamado de servidor, em contra partida um processo cliente contata o servidor, por intermédio da porta lógica que ele está ocupando, a fim de requisitar algum serviço, então o cliente e servidor ficam trocando mensagens, após o cliente ser atendido pelo servidor ele encerra a relação com servidor.

A Figura 4 mostra um código fonte desenvolvido em linguagem Java, onde é criada uma classe cliente, que contém o código para abrir uma conexão com o servidor, enviar uma seqüência de caracteres e receber um retorno do servidor, mostrando-o ao usuário. A linha 09 cria uma conexão com o servidor definido em “hostservername” na porta 6789, e envia uma seqüência de caracteres para o mesmo contido em “sentence+`\n`” na linha14, recebendo o retorno do servidor por intermédio de “inFromServer.readLine()” e armazenando em “modifiedSentence” na linha 15 e mostrando-o na linha 16 e em seguida fechando a conexão na linha 17.

```
01 import java.io.*;
02 import java.net.*;
03
04 class TCPClient{
05     public static void main (String arqv[]) throws Exception{
06
07             String modifiedSentence;
08             BufferedReader inFromUser = new BufferedReader (new InputStreamReader(System.in));
09             Socket clientSocket = new Socket("hostservername",6789);
10             DataOutputStream outToServer = DataOutputStream(clientSocket.getOutputStream());
11             BufferedReader inFromServer = new BufferedReader(new
12 InputStreamReader(clientSocket.getInputStream()));
13             sentence = inFromUser.readLine();
14             outToServer.writeBytes(sentence+`\n`);
15             modifiedSentence = inFromServer.readLine();
16             System.out.println("Resposta: "+modifiedSentence);
17             clienteSocket.close();
18     }
19 }
```

Figura 4. Cliente Socket Java usando a classe TCPClient

Fonte: Franco & Yoshida (2004)

A Figura 5 mostra um código fonte desenvolvido em linguagem Java, onde é criada uma classe servidora, tendo por objetivo receber uma seqüência de caracteres de um cliente por intermédio da porta 6789, transformar esses caracteres em letra maiúscula e enviar novamente ao cliente. Na linha 08 é criada uma escuta na porta 6789 com "ServerSocket(6789)", sendo que na linha 10 é feito o recebimento das informações da porta por "welcomeSocket.accept()", sendo colocado em "inFromClient" na linha 11, sendo colocado em uma variável do tipo caractere na linha 15 e transformado para maiúsculo na linha 16, sendo enviado novamente para o cliente por "outToClient" criado na linha 13 e envia os dados por intermédio da propriedade ".writeBytes" na linha 17.

```
01 import java.io.*;
02 import java.net.*;
03
04 class TCPServer{
05     public static void main(String argv[]) throws Exeption{
06             String clientSentence;
07             String capitalizedSentence;
08             ServerSocket welcomeSocket = new ServerSocket(6789);
09             While(true){
10                     Socket connectionSocket = welcomeSocket.accept();
11                     BufferedReader inFromClient = new BufferedReader( new InputStreamReader
12 (connectionSocket.getInputStream()));
13                     DataOutputStream outToClient = new DataOutputStream(
14 connectionSocket.getOutputStream());
15                     clientSentence = inFromClient.readline();
16                     capitalizedSentence = clienteSentence.toUpperCase()+'\n';
17                     outToClient.writeBytes(capitalizedSentence);
18             }
19     }
20 }
```

Figura 5. Servidor Socket Java utilizando a classe TCPServer

Fonte: Franco & Yoshida (2004)

Além do jogo WAR existe outros jogos que oferecem somente opções multiplayer, ou seja, não existe uma opção de jogo desenvolvendo uma história, sendo exemplos desses jogos o Counter Strike (Sierra) e o Unreal Tournament (Epic Games), alguns, ainda, não oferecem a opção de utilizar jogadores virtuais. Na seção seguinte serão exemplificados alguns jogos que fazem uso de conexão *multiplayer* e que usam jogadores virtuais.

2.3 Outros Jogos

Muitos jogos fazem uso de jogadores virtuais, sendo alguns exemplos os jogos Couter-Strike (Sierra), The Sims (Eletronic Arts), Total Annihilation (CaveDog Entertainment) entre outros. Os jogadores virtuais são usados para diversas finalidades, seja para uma partida em *singleplayer* como em uma partida em *multiplayer*. Em modo *singleplayer* o desenvolver do jogo faz uso desses jogadores virtuais para auxiliar e/ou dificultar a participação do jogador humano na partida a fim de desenvolver a história do jogo e efetivamente chegar ao final do jogo. Esses jogadores virtuais podem ser usados em modo *multiplayer*, sendo que podem ser utilizados, também, para auxiliar ou dificultar a partida e em muitos casos confundido com um jogador humano, tendo reações e tomando decisões como tal.

Alguns jogos são desenvolvidos para executar partidas somente no modo *multiplayer*, ou seja, onde os jogadores interagem sem que haja uma história a se desenvolver, que é o caso do jogo Counter-Strike, Quake 3 arena e Unreal Tournament entre outros. Muitos desses jogos feitos somente para partidas em *multiplayer* não possuem jogadores virtuais, ou seja, todos os participantes são jogadores humanos, sendo que há a possibilidade de incluir jogadores virtuais externos ao jogo como é o caso do jogo Couter-Strike. Em sua maioria, os jogos de tiro em primeira pessoa, como os citados anteriormente, possibilitam a entrada de jogadores no decorrer da partida, ou seja, não há a necessidade de que haja uma quantidade mínima de jogadores para iniciar uma partida, mas em alguns jogos os jogadores devem estar presentes antes de a partida ser iniciada, sendo que após o seu início não há a possibilidade ser adicionado jogadores enquanto esta não chega ao término.

O WAR é um jogo onde só pode ser iniciada uma partida com o mínimo de três jogadores, sendo que todos devem estar presentes desde o início da partida, e não podendo ser incluído nenhum jogador no decorrer da mesma. Cada jogo possui características específicas, onde foram avaliadas as seguintes características (Tabela 2):

- Tipo de Jogo: onde é avaliado o jogo conforme o gênero;
- Tipo de Jogador Virtual: onde é identificado se os jogadores são implementados no jogo (interno), ou se os jogadores virtuais devem ser acrescentados ao software, sendo eles da própria empresa desenvolvedora ou de terceiros (externo); e
- Incluir Jogadores: indica como os jogadores podem ser incluídos na partida, se é somente no início ou se durante o desenrolar do jogo podem ser incluídos mais jogadores.

Tabela 2. Características de Outros Jogos X WAR

	Total Annihilation	**Counter-Strike**	**Unreal Tournament**	**WAR (Projeto)**
Tipo Jogo	Estratégia	Tiro em Primeira Pessoa	Tiro em Primeira Pessoa	Estratégia
Tipo Jogador Virtual	Interno	Externo	Interno	Interno
Incluir Jogadores	Início Partida	Início e Durante a Partida	Início e Durante a Partida	Início Partida

Normalmente os jogos do gênero de estratégia exigem que os jogadores estejam presentes no início da partida, como é o caso do WAR, onde há a distribuição dos territórios, e após esta distribuição não há espaço para a inclusão de um novo jogador que tenha eventualmente chegado no meio da partida.

Alguns jogos utilizam a Inteligência Artificial para dar maior realismo para os personagens, e maior emoção ao jogo. A seguir serão apresentados os conceitos de Inteligência Artificial.

2.4 Inteligência Artificial

Conforme Lustosa (2004), a partir da filosofia começou o estudo da inteligência e dos processos de aprendizagem, onde estudiosos gregos já produziam textos sobre o tema. Sócrates, em 400 a.C., testou o conhecimento das pessoas afim de saber quem era a mais sábia do mundo. Hoje em dia os testes de Quoeficiente de Inteligência (QI) tentam entender e categorizar a inteligência humana.

Ainda conforme Lustosa (2004), no final do século XIX os pesquisadores aprofundaram os seus estudos de forma mais sistemática, embasando-se na psicologia, educação, sociologia, medicina entre outras. Nestes cem anos foram apresentadas muitas teorias sobre o funcionamento

da mente humana, muitas baseadas em fatores fisiológicos, passando pelas baseadas em estímulos externos.

Em geral, os dicionários da língua portuguesa como Michaelis (1998), definem inteligência como sendo “a faculdade de aprender, compreender e adaptar-se" e apresentam pelo menos mais três acepções distintas para esse termo:

- Teológica: “dom divino que nos torna semelhantes ao Criador";
- Filosófica: “princípio abstrato que é a fonte de toda a intelectualidade"; e
- Psicológica: “capacidade de resolver problemas novos com rapidez e êxito".

Existem muitas definições sobre inteligência, como segue:

- "Inteligência é julgar bem, compreender bem, raciocinar bem" (BINET, 1929);
- Humberto Maturana (uma das abordagens): "Representa um atributo ou propriedade distintiva de alguns organismos" (MATURANA, 1998); e
- "Capacidade de adaptação do organismo a uma situação nova" (PIAGET, 1982).

“A Inteligência Artificial (IA) tenta entender o comportamento de entidades inteligentes. Mas, ao contrário da filosofia e da psicologia, que estão mais preocupadas com o estudo da inteligência dentro de um contexto de relações humanas, a IA é mais focada em como essas entidades podem ser criadas e utilizadas para determinados fins” (LUSTOSA, 2004).

A inteligência é a demonstração por resultados de princípios coerentes, em escala de tempo verificável: a natureza é inteligente em escala de tempo inacessível para os humanos. O oposto da inteligência é o caos: em um sistema caótico, duas entradas muito próximas resultam em duas saídas sem qualquer conexão (CONAI, 1994).

Para McCarthy (2002), inteligência é a parte computacional da habilidade de alcançar objetivos, percebendo-se diversos tipos e graus de inteligência em pessoas, em muitos animais e em máquinas.

Há uma grande divergência quanto ao significado de inteligência e, dessa forma, definir precisamente o que é inteligência artificial é uma tarefa, se não impossível, pelo menos extremamente difícil. Entretanto, podemos definir Inteligência Artificial (IA), enquanto disciplina do conhecimento humano.

Segundo Russell & Norvig (1995), as definições de IA, encontradas na literatura científica, podem ser agrupadas em quatro categorias:

- Sistemas que pensam como humanos;
- Sistemas que agem como humanos;
- Sistemas que pensam corretamente (de acordo com as leis da Lógica); e
- Sistemas que agem corretamente (de acordo com as leis da Lógica).

Segundo Pereira (2003) as duas primeiras categorias são essencialmente empíricas e envolvem formulação de hipóteses e confirmação experimental, enquanto as outras duas são teóricas e envolvem matemática e engenharia. Embora essas duas linhas de pesquisa muitas vezes pareçam antagônicas, os resultados obtidos em ambas têm contribuído bastante para o desenvolvimento da área de IA.

A seguir têm-se algumas definições de IA:

- "A arte da criação de máquinas que fazem funções que requerem inteligência quando feito por pessoas" (KURZWEIL, 1990);
- "Campo de estudo que procura explicar e emular comportamento inteligente em termos de processos computacionais" (SCHALKOFF, 1990);
- "Um ramo da ciência da computação que se dedica à automação de comportamento inteligente" (LUGER & STUBBLEFIELD, 1993); e
- "A parte da Ciência da Computação que compreende o projeto de sistemas computacionais que exibam características associadas, quando presentes no comportamento humano, à inteligência" (BARR & FEIGENBAUM, 1981).

Segundo Lustosa (2004), o termo Inteligência Artificial surgiu em 1956, de uma reunião feita nos Estados Unidos que contou com cientistas de várias áreas do conhecimento, todos interessados no mesmo objetivo, no estudo de como poderiam ser criadas máquinas inteligentes, e de como estas afetariam os processos existentes.

Percebe-se, que cada especialista na área de IA aponta uma explicação para a mesma. Segundo Arariboia (1988), a Inteligência Artificial é um campo que usa técnicas de programação que procuram, por intermédio de máquinas, resolver problemas do mesmo modo que um ser

humano os resolveria. Já para McCarthy (2002), a Inteligência Artificial é a ciência e a tecnologia de construção de máquinas inteligentes, especialmente programas de computador.

A inteligência é difícil de definir, podendo diferir conforme o ponto de vista de cada autor, e sendo a IA uma área que tenta recriar a inteligência de maneira sintética, também tem uma variedade de definições, como também faz uso de inúmeras técnicas para alcançar o seu objetivo. A Inteligência Humana caracteriza-se por "funções psicológicas ou conjuntos de funções graças às quais o organismo se adapta ao seu meio produzindo combinações originais de condutas, adquire e explora conhecimentos novos e, eventualmente, raciocina e resolve os problemas de uma maneira conforme às regras destacadas pelas formalizações da lógica" Doron & Parot, (2001). E conforme Lino (2004), dito por outras palavras, a inteligência é um conjunto de funções que o homem inatamente possui, que ao longo da vida vai desenvolvendo e crescendo, que o ajudam a adaptar-se ao meio em que vive bem como a tomar decisões e a escolher perante os obstáculos que se apresentam. O raciocínio, a memória, a linguagem e as emoções são componentes da inteligência humana, sem os quais o humano não se tornaria autônomo e apto a sobreviver.

E ainda defende Lino (2004), que a Inteligência Artificial por seu lado é uma nova ciência, que "bebe" teorias tanto da informática como das experiências da psicologia cognitiva, tendo como objetivo geral reproduzir por meio de máquinas, nomeadamente computadores, atividades humanas qualificadas como inteligentes. Cabe assim à Inteligência Artificial o tratamento de informações simbólicas não numéricas utilizando heurísticas, a resolução geral de problemas, o reconhecimento de formas e o reconhecimento automático da linguagem natural. De fato, a Inteligência Artificial assemelha-se em muitas coisas à inteligência humana, pois outros dos seus domínios é o desenvolvimento de sistemas à base de representação de conhecimento e a modelização de raciocínios. Assim, tal como a inteligência humana, também a Inteligência Artificial utiliza o raciocínio lógico, a memória e a linguagem, mas falta-lhe um componente bastante importante da inteligência humana, a emoção. Na secção de Sistemas Especialistas será apresentada uma das inúmeras técnicas existentes para representar o conhecimento humano e que será utilizada no desenvolver deste projeto.

2.4.1 Sistemas Especialistas

Quibel e Roger (1989) iniciam sua definição da engenharia do conhecimento pela definição da própria palavra engenharia: "A engenharia comporta os métodos para estudo de um projeto e realização de um trabalho complexo por equipes diversas, de especialidades complementares", e

continua dizendo que "desenvolver um SE é aplicar uma "engenharia" do conhecimento: a construção de uma informática lógica, refere-se à fatos e idéias e a capacidade de tratá-los, os quais receberá do exterior, para deduzir um diagnóstico, uma preconização, ou, de fato, uma decisão".

E conforme Crippa (2004) os SE têm sua origem no fim da Segunda Grande Guerra. Nessa época, grupos independentes de cientistas ingleses e norte-americanos trabalhavam numa máquina eletrônica que se pudesse conduzir por um programa armazenado de instruções e fosse feita para executar cálculos numéricos complexos, o que hoje se chamaria computador.

E segundo Flores (2003), o conhecimento humano provém de tarefas que são realizadas tendo elas sucesso ou não, e que esses conhecimentos adquiridos são armazenados em forma de regras práticas a fim de utilizá-las em um momento oportuno.

A modelagem de um sistema computacional para resolver problemas que utilizam conhecimento humano, chamados de Sistemas Baseados em Conhecimento (SBC), requer a captura dessas regras (heurísticas) em um processo chamado de aquisição de conhecimento. Esse processo torna-se o mais difícil em um projeto de SBC, tendo em vista que o conhecimento humano é a soma de toda a experiência de vida da pessoa (FLORES, 2003)

Conforme Rezende (2003), os SBC vêem sendo utilizados a mais de 20 anos, a fim de resolver problemas que, de forma computacional seria complexa e, na maioria das vezes, requer a presença de um especialista humano para resolvê-lo.

Flores (2003), diz que um SBC chega ao status de SE quando sua base de conhecimento tenha maturado a um status especialista.

Segundo Feigenbaum (1988) (*apud* HARMON e KING, 1988), um SE é um programa para resolver problemas difíceis e para tal utiliza o conhecimento e procedimentos que são inerentes aos melhores profissionais do ramo. Esse conhecimento provê de fatos heurísticos, que por sua vez constituem um corpo de informação que é largamente compartilhado, publicamente disponível e geralmente aceito pelos especialistas em um campo. Já as heurísticas na maioria das vezes são privadas e inerentes, que caracterizam a tomada de decisão em nível de especialista na área.

Conforme afirma Sellitto (2002),

> "Sistemas especialistas são sistemas computacionais que resolvem problemas de forma similar ao modo como um especialista humano os resolveria, possuindo capacidade de

> decisão em campos específicos do conhecimento. Um sistema especialista resolve problemas em uma área limitada de conhecimento, não se devendo esperar do mesmo que possa ter capacidade de resolver qualquer tipo de problema".

Flores (2003), explica que um SE é formado basicamente por: base de conhecimento, mecanismo de inferência, explanação e aquisição de conhecimento (Figura 6).

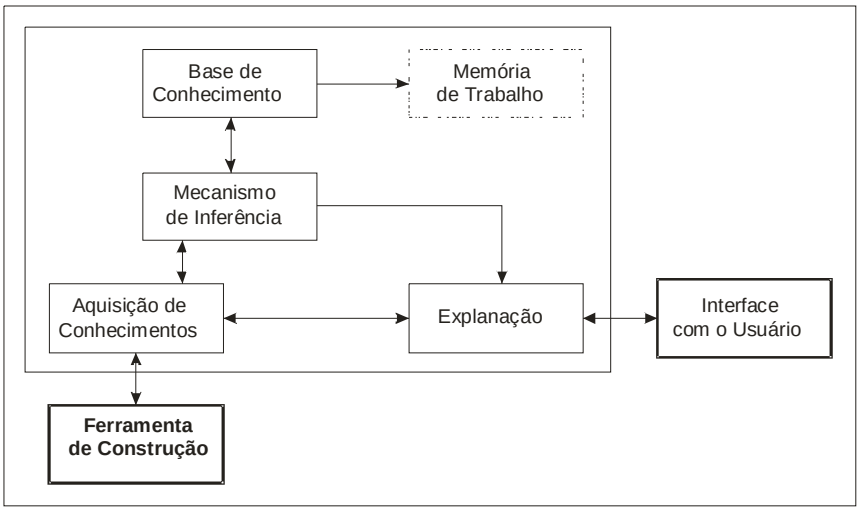

Figura 6. Arquitetura básica dos sistemas especialistas.

Fonte: Flores (2003).

A base de conhecimento representa o conhecimento do especialista. O mecanismo de inferência é responsável pela busca na base do conhecimento e transfere os fatos e regras utilizadas durante a consulta para a memória de trabalho. O módulo de aquisição do conhecimento é responsável pela atualização da base de conhecimento, vinculado a ele está a ferramenta de construção, através da qual, o sistema permite a criação de novos SE. O módulo de explanação é responsável pela descrição do raciocínio do sistema para o usuário. Pode ser executado a partir do mecanismo de inferência visando um processo de transformação do conhecimento representado na base de conhecimento ou deduzido no processo de busca de uma solução, como também pode ser executado pelo módulo de aquisição de conhecimento, auxiliando na depuração e construção incremental da base de conhecimento.

Para implementar a base de conhecimento pode ser utilizada uma ferramenta de inferência, que permite a inclusão das e a consulta na base. Na próxima seção é identificada uma ferramenta de inferência a ser utilizada no projeto.

2.4.2 Ferramentas de Inferência

As regras determinam como o jogador virtual reage ao ambiente, sendo elas implementadas utilizando uma estrutura de árvore de decisão. Para representar esta estrutura de árvore, existem linguagens que facilitam sua implementação como o CLIPS (C Language Integrated Production System).

Conforme Riley (2007), CLIPS foi desenvolvido NASA (National Aeronautics and Space Admistration) afim de facilitar o desenvolvimento de sistemas especialistas. Inicialmente o CLIPS interage com linguagens como o C e o FORTRAN. Para que possa ser utilizado com outras linguagens há a necessidade de se utilizar uma API, que faz a conexão com o CLIPS e a linguagem ao qual deseja interagir.

Para interagir com o Java existem duas API´s, sendo elas o JESS e o JCLIPS. O JESS é produzido pela Sandia National Laboratories, e possui uma licença para uso comercial, sendo sem custos para uso acadêmico. Já o JCLIPS é de domínio público, sem custos de licença, mas não permite mais do que uma consulta a base de conhecimento por vez.

Para o projeto será utilizado o JCLIPS por se tratar de uma ferramenta livre de licença, tanto para desenvolvimento quanto para distribuição, sendo que o sistema não precisará fazer mais do que uma consulta por vez na base de conhecimento, não necessitando de uma ferramenta que permita isso.

2.4.3 Agentes Inteligentes

Segundo Lucena (2003), através dos tempos, a humanidade preocupou-se muito em desenvolver a área da ficção e da engenharia, tentado criar agentes inteligentes que pudessem imitar o comportamento humano. Com esse início a área da Inteligência Artificial acreditou ser possível iniciar as pesquisas dos mesmos, e assim inventaram os primeiros agentes, alguns bem conhecidos como: secretárias eletrônicas, brinquedos que tocavam sozinhos algum tipo de música, trenzinhos eletrônicos, etc. E como cita Bradshaw (1996), por anos seguidos, foram criados personagens como andróides, cyborgs, robôs, humanóides e computadores inteligentes para serem usados em filmes que mostravam guerra no espaço, com direito a robôs como guias, soldados, etc; assim eram vistos como figuras de um futuro próximo.

Para Clark (1968), mostrar que a era da inteligência estava aflorando era muito importante e foi através do filme em 1968 “2001 *a space odyssey*”, que comprovou através do computador HAL 9000, que futuramente os computadores teriam emoções, atitudes e tomariam decisões próprias. Conforme Turing (1950), o teste de Turing, veio propor que máquinas seriam inteligentes se um ser humano não pudesse distinguir o comportamento de uma máquina de outro ser humano. Essencialmente, o teste proposto por Turing era que, por meio de um terminal, um ser humano deveria se comunicar com “alguém” num local remoto (como num *Chat*); se, após um determinado tempo, ele não fosse capaz de perceber que se esse “alguém" era uma máquina, então a hipótese da existência de Inteligência Artificial estaria confirmada.

Conforme diz Reingold e Nightingale (1999), com vontade de realizar o sonho de ter máquinas imitando o ser humano, foi na década de 1950, que um grupo de criativos e jovens cientistas se reuniram no Dartmouth Summer Reseach Project para discutir uma nova e revolucionária idéia: como construir máquinas inteligentes. O grupo era composto por Marvin Minsky, Herbert Simon, Allen Newell e John McCarthy, entre outros, onde seus esforços abririam portas para desvendar o projeto de máquinas inteligentes. Newell e Simon apresentaram no encontro de 1956 um programa chamado LT, o *Logic Theorist*. O programa era revolucionário para a época, pois conseguia provar certos teoremas da aritmética. Esse foi um dos primeiros programas que foram empregados na manipulação de informação não numérica. E conforme Newell & Simon (1976) “Um sistema simbólico físico tem os meios necessários e suficientes para a ação inteligente genérica".

Mas, no início não tiveram o êxito esperado, sendo que continuaram as suas buscas através dos anos. E segundo Negroponte (1995), foi na década de 1990, que iniciaram a suas atividades no campo dos agentes inteligentes, na área de computação autônoma. Com a explosão da Internet na rede *World Wibe Web*, em 90 que surgiu à necessidade de desenvolver agentes inteligentes para auxiliar os usuários da rede, onde os mesmos poderiam ajudar em sites de consultas, jogos, pesquisas, processamento de informações mais rápidas e conseqüentemente agilizando e aproveitando os benefícios da rede.

E através dos estudos e ponto de vista de cada pesquisador, criaram várias definições para o termo agente como forma de entendimento do processo e aspectos que envolvam o termo.

Segundo Bax (2001), embora não haja uma definição universal para o termo “agente”, no âmbito da computação, pode-se considerar o conceito disseminado de agentes como assistentes de

tarefa, ou seja, entidades de software que empregam técnicas de Inteligência Artificial com o objetivo de assistir ao usuário na realização de uma determinada tarefa, agindo de forma autônoma e utilizando a metáfora de um assistente pessoal.

E como cita Costa (1999), o termo agente é utilizado na literatura computacional para determinar diversos tipos de programas, não necessitando apresentar comportamento inteligente.

Sabe-se que, dado um determinado sistema, denomina-se agente cada uma de suas entidades ativas. “O conjunto de agentes neste ambiente constitui uma sociedade. Tal ambiente é formado pelas entidades passivas. A um agente cabe “raciocinar” sobre o ambiente, sobre outros agentes bem como decidir racionalmente objetivos a seguir e ações a tomar” (FERREIRA, 1993).

Uma definição segundo Maes (1995) (apud Costa, 1999), diz que agentes são componentes de software que atuam automaticamente de forma que atendam os interesses dos usuários.

E segundo Heidlmann *et al* (apud Faraco, 1998), diz informalmente que agentes são tidos com alguém ou alguma coisa que age como procurador com o propósito especifico de realizar ações, tarefas que podem ser compreendidas como benéficas a parte que esta representando.

“Um agente é um sistema de computador que está situado em algum ambiente e que é capaz de executar ações autônomas de forma flexível neste ambiente, a fim de satisfazer seus objetivos de projeto” (WOOLDRIDGE, 1999).

> Um agente é uma entidade à qual se pode associar uma identidade única, e que é capaz de realizar cálculos formais. Um agente pode ser considerado como um meio que produz um certo número de ações a partir dos conhecimentos e mecanismos internos que lhe são próprios (GASSER, 2002).

“Um agente é um software que sabe fazer coisas que provavelmente você faria você mesmo se tivesse tempo” (SELKER, 2001).

Uma definição associada a agentes em software, capazes de trafegar através de uma rede de computadores e interagir com outros elementos autônomos, é: "um agente é uma entidade que executa um conjunto de operações que lhes foram incumbidas por um usuário ou outro programa, com algum grau de independência ou autonomia e, executando estas operações, emprega algum conhecimento dos objetivos ou desejos do usuário" (IBM, 1998).

A “flexibilidade” das ações é o que distingue um agente de um programa de controle de processos. Segundo Jennings *et al* (1998), flexibilidade significa que o sistema é:

- Reativo: agentes devem perceber seu ambiente e responder oportunamente às mudanças que ocorrem nele;
- Pró-ativo: agentes não devem simplesmente atuar em resposta ao ambiente, devem exibir um comportamento oportunista e direcionado ao seu objetivo e tomar a iniciativa quando apropriado; e
- Social: agentes devem interagir, quando apropriado, com outros agentes artificiais ou humanos para completar suas próprias soluções de problemas ou ajudar outros com suas atividades.

Segundo Wooldridge (1999) as três características citadas, reativo, pró-ativo e social, são suficientes para classificar um agente como inteligente. Os agentes inteligentes, mais que todos os outros ramos da IA, estão presentes na resolução dos problemas dos usuários comuns. Hoje, a internet conta com diversas iniciativas que utilizam agentes, desde sites que comparam preços de produtos para compra até mecanismos de busca inteligentes, que navegam dentro das páginas *web*, apresentando o resultado da busca classificado pelo grau de acerto e relevância dos assuntos.

A Figura 7 ilustra as características de um agente sob a perspectiva de Wooldridge e Jennings (1998). São evidenciadas as capacidades de persistência (estado interno), reatividade (percepções e ações) e pró-atividade (ações).

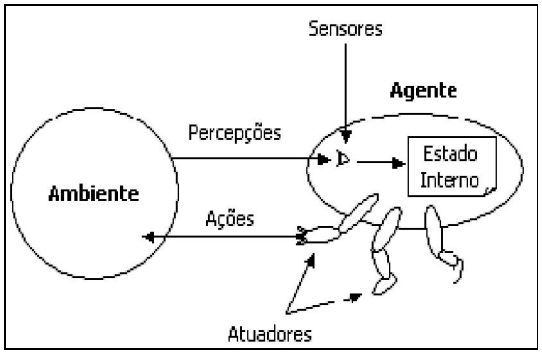

Figura 7. Características de Agentes
Fonte: Grimshaw (2001)

Um agente deve ser capaz de perceber seu ambiente através de sensores e agir de forma autônoma sobre este ambiente através de seus executores que realizam ações. (RUSSEL, 1995).

Contudo Russel e Norvig (1995), deixa em aberto a definição da racionalidade em si, o que não esclarece por completo o tipo de comportamentos esperados dos agentes para que eles possam ser considerados inteligentes.

No entanto a inteligência dos agentes sempre foi motivo de intensivas pesquisas no ramo de IA e também no que diz respeito a outras faculdades como a filosofia e a psicologia, onde não se pode dizer com precisão se a inteligência apresentada nos agentes é a mesma inteligência vista nos seres humanos. (MINSKY, 2000). Esses organismos inteligentes têm uma especial atenção ao nível perceptual, aquele nível que consegue, por exemplo, discriminar uma maçã de um tomate. Inteligência requer, também, o equivalente de um "corpo físico" que permita ao agente explorar suas circunvizinhanças e assim obter uma oportunidade de interação (CLARK, 1997).

Por isso, alguns pesquisadores como Nissen (1995), iniciaram trabalhos com intuito de esclarecerem os comportamentos dos agentes e classificá-los. E para tanto, fez-se uma relação de tópicos essenciais a esses agentes, podendo assim conhecer o seu comportamento a seguir:

- Autonomia: é a capacidade em que o agente tem de tomar decisões sozinhos, sem interferências dos usuários. E conforme Wooldridge & Jennings (1994) **(***apud* Costa, 1999), o elemento chave da autonomia é a pró - atividade, habilidade de tomar iniciativa, sem a necessidade de agir a mudança de seu ambiente;
- Capacidade de comunicação: ela é necessária para um agente conseguir atingir seus objetivos, podendo se comunicar com outros agentes ou com o usuário. Para Franklin *et al* (1990) (*apud* Faraco, 1998), é a capacidade de comunicação do agente com outros e as pessoas;
- Capacidade de Cooperação: agentes podem se comunicar entre si, formando uma comunidade de agentes a fim de resolver um problema; está ligada a Inteligência Distribuída, uma das primeiras áreas que fizeram usos dos agentes. Para Nwana (1996) (*apud* Costa, 1999), acredita que a cooperação é fundamental, para a existência de um ambiente multiagente;
- Capacidade de Raciocínio: o conhecimento dos agentes pode ser expresso simbolicamente ou através de dados processados por meio da IA de formas diferentes, como: redes neurais, algoritmos genéticos, etc;

- Planejamento: para atingir metas os agentes precisam de planejamento, como HTN (Hierarchical Task Networks), entre outros; e
- Adaptabilidade: analisar o ambiente e reagir às situações que possam ocorrer.

Como visto, pode-se verificar a existência de várias definições para o termo agente e suas características em comum, mas também faz-se necessário a classificação dos mesmos, para um entendimento maior. Conforme Giese (1998), os agentes podem ser classificados de diversas formas como: nível de inteligência, tarefas que executam, quanto à mobilidade, quanto à aquisição de inteligência e a ênfase dada a alguns atributos primários.

O primeiro deles, diz respeito quanto ao Nível de Inteligência, segundo River (1994) (*apud* Souza, 1996), são classificados da seguinte forma:

- Baixo: neste estágio, executam tarefas rotineiras, não se adaptam a mudanças, não alteram suas regras;
- Médio: estes agentes utilizam-se de uma base de conhecimento para desempenharem suas funções. Podem adaptar-se a mudanças e manipular novas condições, mas não desenvolvem novas regras; e
- Alto: os softwares de agentes utilizam o aprendizado quanto à base de conhecimento. Aprendem com o comportamento dos usuários. Criam novas regras e mudanças de condições.

Outro fator de classificação de agentes diz respeito aos tipos de tarefas que esses agentes executam. Wooldridge e Jennings (1995) (*apud* Souza, 1996), afirmam ter três classes de agentes segundo a tarefa que estiverem executando, então:

- Gopher: termo dos agentes que executam tarefas simples, baseando-se em suposição. Ex: agenda eletrônica e de compromissos;
- Prestadores de Serviço: são aqueles agentes que executam tarefas definidas, quando solicitadas pelo usuário. Eles podem negociar datas com os usuários; e
- Pró-ativo/preditivo: esses agentes efetuam as tarefas mais difíceis, apresentando informações voluntárias para o usuário, sem ser solicitado, ou seja, faz isso quando achar necessário.

E de acordo com Nwana (1996) (*apud* Giese, 1998), os agentes são classificados também pela sua Mobilidade que segue dois itens:

- Agentes Estáticos: são agentes que não se locomovem pela rede, estão fixo em um lugar; e
- Agentes Móveis: tem habilidade de locomoção pela rede.

E outro tópico diz respeito quanto a sua Aquisição de Inteligência, onde Nwana (1996) (*apud* Giese, 1998), afirma que eles podem ser classificados a partir dos passos abaixo:

- Deliberativos: os agentes possuem um modelo simbólico de raciocínio interno, planejam e negociam tarefas para alcançar coordenação com outros agentes. Eles prevêem que ações serão necessárias para alcançar o objetivo, para então executá-las; e
- Reflexivos/ Reativos: esses têm um mapeamento de situações e respostas associadas. Utilizam o estímulo – resposta.

O último tópico fala do Quanto à Ênfase dada a Alguns Atributos Primários, e conforme diz Nwana (1996) (*apud* Kern, 1998), que sucedem (cooperação, aprendizado e autonomia), e ainda afirma que estas etapas não são definitivas.

No presente projeto os agentes foram implementados como jogadores virtuais. A Tabela 3 mostra como os jogadores virtuais atendem as características de um agente inteligente.

Tabela 3. Agente Inteligente X Jogadores Virtuais

Características de Um Agente Inteligente	**Jogadores Virtuais**
Reativo: Interação com o ambiente	Analisar o estado atual do jogo e executar ações baseado nesta análise
Pró-Ativo: Independente e orientado a um objetivo	Atacar e defender os territórios conforme o objetivo
Social: Interação com outros agentes ou humanos	Possibilidade de criar alianças com outros jogadores

Os agentes inteligentes devem ter um certo grau de comunicação seja entre eles, com o sistema ou mesmo com os jogadores humanos, na secção a seguir serão apresentadas técnicas para comunicação de agentes inteligentes.

2.4.4 Comunicação entre Agentes

Da mesma maneira que os seres humanos precisam interagir com as demais pessoas que as cercam a fim de realizar uma tarefa, a mesma necessidade os agentes sentem na hora de se comunicar para atingir um objetivo. E essa situação inclui: consultar outros agentes; responder a perguntas de outros agentes participantes; aceitar sugestões, propostas, desafios e impedimentos; dividir as experiências vividas no meio virtual, etc. (LUCENA, 2003). E conforme Loh e Garin (2004), quando existem vários agentes inteligentes atuando de forma integrada e cooperativa, o sistema chama-se de sistema de multiagentes. Geralmente, cada agente inteligente possui conhecimentos próprios e diferentes. Estes indivíduos interagem entre si, compartilhando informações e conhecimento para soluções de problemas mais complexos, os quais dificilmente seriam resolvidos por qualquer um dos indivíduos de maneira isolada.

Para tanto, os agentes possuem meios de comunicação entre eles e cada modelo tem uma característica para a troca de informação, que pode ser o uso do quadro-negro ou a troca de mensagens, como: páginas amarelas e páginas brancas.

O quadro-negro, segundo Martin *et al* (1999), é uma estrutura de dados persistente onde existe uma divisão em níveis, visando facilitar a busca de informações. É por ali que os agentes se comunicam, interagem. E conforme Lemos *et al* (2002), os agentes que necessitam de alguma informação escrevem seu pedido no quadro à espera que outro agente acesse e responda.

Segundo Lucena (2003) a comunicação por troca de mensagens, consiste na comunicação direta entre os agentes. Por isso faz-se necessário que cada agente saiba os nomes e endereços de todos os agentes que formam o sistema para que as mensagens possam ser trocadas. E segundo Helin e Laukkanen (2002), a simplificação da comunicação é feita através de agentes facilitadores que armazenam os endereços e serviços oferecidos pelos agentes do sistema. Nesse caso, quando um agente necessita se comunicar com outro agente, esse pode solicitar que o agente facilitador encontre o agente procurado pelo nome (através das páginas brancas – *pages White*) ou pelo tipo de agente ou serviço oferecido (através das páginas amarelas – *pages Yellow*). E segundo Helin e Laukkanen (2002), os agentes podem também acessar os serviços diretamente das páginas amarelas ou brancas sem a necessidade de agentes facilitadores.

Sendo assim, é preciso algum meio de comunicação para a troca de mensagens através das páginas Brancas e Amarelas. Os agentes precisam de uma linguagem para representação das

mensagens, como também, uma linguagem de representação do conhecimento (conteúdo), codificação das mensagens e um meio para transportar essa mensagem.

Para um software ser definido como agente basta implementar as características que definem um agente. Dessa forma são gerados agentes com padrões próprios de cada desenvolvedor, sendo assim dificultando a interação entre agentes de diferentes desenvolvedores. A FIPA (Foundation for Intelligent Physical Agents) define padrões para a implementação de agentes, e facilitando a comunicação entre eles.

2.4.5 FIPA

Conforme Silva (2003), a FIPA é uma fundação sem fins lucrativos para a padronização de agentes heterogêneos e interativos, mas que também padroniza agentes robóticos. Foi fundada em 1996 em Genebra, com a missão oficial Fipa (2003): "A promoção de tecnologia e especificações de interoperabilidade que facilitem a comunicação entre sistemas de agentes inteligentes no contexto comercial e industrial moderno". E para essa equipe formalizar todos os critérios é necessário que seja composta por companhias e organizações membro que se encontram em reuniões para discutir e produzir padrões para a tecnologia de agentes. E conforme Fipa (2003) segue a divisão desse comitê e suas funções:

- **O Comitê Executivo** é responsável por executar as atividades a ele confiadas pela Reunião Plenária do FIPA. O Comitê Executivo é composto pelo Presidente do FIPA, dois representantes de cada uma das sub-regiões e um representante do país anfitrião da próxima Reunião Plenária;
- **Comitês Técnicos** *(Technical Comittees* - TC) – Essa equipe fica responsável pela parte do trabalho técnico, criando ou alterando as especificações existentes. Atuam na arquitetura do agente, sua descrição, localização, permissões e obrigações, e também na conformidade que são as especificações da FIPA;
- **Grupos de trabalho** (*Working Groups* - WG) – Essa equipe divulga informativos ou propostas de alterações às especificações existentes na Fipa;
- **Comissão de diretores** (*Board of Directors* - BD) – Responsáveis pela gestão da FIPA, definindo políticas e procedimentos que a FIPA adota para os ambientes de desenvolvimentos de agentes; e

- **Comissão de arquitetura** (*Architecture Board* -AB) - Responsáveis por assegurar a consistência e precisão do trabalho técnico a ser desenvolvido.

Então, conforme Fipa (2003) antes dessa padronização havia cerca de 60 sistemas proprietários de agentes em competição, mas não tinha padronização atrasando o desenvolvimento da tecnologia de agentes. Por isso, a utilização de um padrão julgava-se de suma importância na área. Algumas das linguagens de desenvolvimento de agentes que estão relacionadas com a FIPA são: Agent Development Kit[2], FIPA-OS, Lightweight Extensible Agent Plataform[3] (LEAP), ZEUS e o JADE.

O JADE é um conjunto de classes e funcionalidades, chamado de *framework,* desenvolvido em Java e que implementa os padrões da FIPA, afim de facilitar a implementação de agentes inteligentes em Java.

2.4.6 JADE

O JADE (*Java Agent DEvelopment framework)* é uma estrutura de desenvolvimento de software que auxilia no desenvolvimento de agentes, e que entra em conformidade com os padrões de FIPA (Foundation for Intelligent Physical Agents). Foi desenvolvido e suportado pelo CSELT da Universidade de Parma na Itália, é *open source* ,desenvolvida sob a licença LGPL (Lesser General Public License), permitindo que possa ser ligada a programas proprietários, inclui dois produtos principais: uma plataforma FIPA-*compliant* do agente e um pacote para desenvolver agentes em Java. De acordo com Bellifemine (2003), o JADE foi codificado em Java devido a características específicas da liguagem, como a programação orientada a objeto em ambientes heterogêneos.

2.4.6.1 Característica do JADE

No JADE, existem algumas características principais para a programação de multiagentes que serão abordadas a seguir de acordo com Bellifemine *et al* (2003):

- **Plataforma distribuída de agentes**: JADE pode ser dividida em vários "*hosts*" ou máquinas (desde que conectados via RMI). Os agentes são implementados como *threads* Java e inseridos dentro de repositórios de agentes chamados de containeres (*Agent Containers)* que provêm todo o suporte para a execução do agente;
- **GUI ou *Graphical User Interface*:** Interface visual que gerencia vários agentes e containeres de agentes inclusive remotamente;

- **Ferramentas de *Debugging*:** ferramentas para o desenvolvimento e depuração de aplicações multiagentes baseados em JADE;
- **Suporte a execução de múltiplas, paralelas e concorrentes atividades de agentes**: através dos modelos de comportamentos (*Behaviours*);
- **Ambiente de agentes complacentes a FIPA**: No qual incluem o sistema gerenciador de agentes (AMS - *Agent Management System*), o diretório facilitador (DF - *Directory Facilitator*) e o canal de comunicação dos agentes (ACC - *Agent Communication Channel*). Todos esses três componentes são automaticamente carregados quando o ambiente é iniciado;
- **Transporte de mensagens**: Transporte de mensagens no formato FIPA-ACL dentro da mesma plataforma de agentes;
- **Biblioteca de protocolos FIPA** - Para interação entre agentes JADE dispõe uma biblioteca de protocolos prontos para ser usados;
- **Automação de registros**: Registro e cancelamento automático de agentes com o AMS fazendo com que o desenvolvedor se abstraia disso;
- **Serviços de nomes (*Naming Service*) em conformidade aos padrões FIPA**: na inicialização dos agentes, estes obtêm seus GUID (*Globally Unique Identifier*) da plataforma que são identificadores únicos em todo o ambiente; e
- **Integração**: Mecanismo que permite aplicações externas carregarem agentes autônomos JADE.

2.4.6.2 Arquitetura Interna do JADE.

A base da arquitetura da plataforma JADE está na existência de várias máquinas virtuais Java (JVM) sendo distribuída por diversas máquinas independente de sistema operacional que cada uma utiliza.

Conforme SILVA (2003) a Figura 8 demonstrando a distribuição da plataforma em 3 *host.s* Sendo que a JRE 1.2 (*Java Run-time Enviroment,* versão 1.2) para enfatizar o conceito de independência de plataforma. A Figura 8 mostra que, a estrutura de comunicação entre agentes do JADE é transparente ao desenvolvedor, e que seus containeres fornecem todo o suporte para o funcionamento de um ou mais agentes por máquina. A distribuição dos containeres se utiliza de um

container principal, definido como Main Container, onde está o RMI, que nada mais é que um servidor de nomes usado para registrar e recuperar referências de objetos através do nome, e é onde se encontram também o AMS e o DF. Esse RMI é a forma que JADE utiliza para manter os containeres de agentes que se conectam a plataforma. O Container de Agentes JADE fornece um ambiente completo para a execução dos agentes em um computador de forma concorrente.

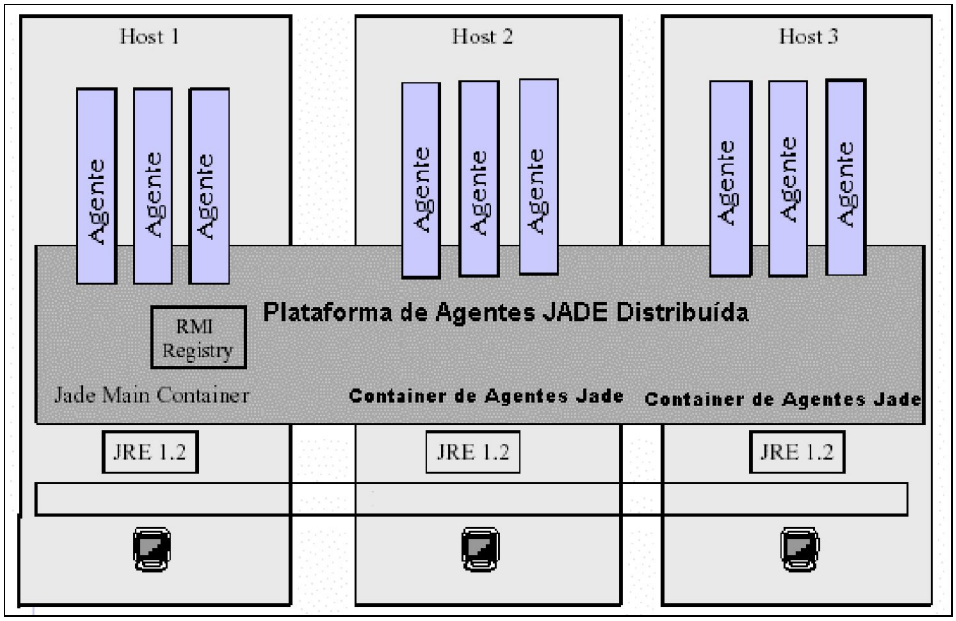

Figura 8. Plataforma de agente JADE

Fonte: Silva (2003)

2.4.6.3 Especificações dos Agentes em JADE

Para o JADE, um agente é autônomo e um processo independente que tem uma identidade e requer comunicação com outros agentes, seja ela por colaboração ou por competição, para executar totalmente seus objetivos (JADE, 2003). Em outros termos o agente de JADE emprega múltiplas funções ou comportamentos, ele é implementado na classe chamada *Agent* no pacote "jade.core." Onde estão definidos tarefas básicas como: passagens de informações (mensagens), suporte no ciclo de vida dos agentes, escalonamento e interação com sistemas de agentes da FIPA.

Na Figura 9 há uma descrição da arquitetura interna de um agente genérico em JADE. Na parte superior os comportamentos ativos do agente que representariam as ações e/ou intenções que cada agente tem. O modelo computacional de um agente em JADE é multitarefa, onde tarefas (ou

comportamentos) são executadas concorrentemente. Cada funcionalidade ou serviço provido por um agente deve ser implementado como um ou mais comportamentos.

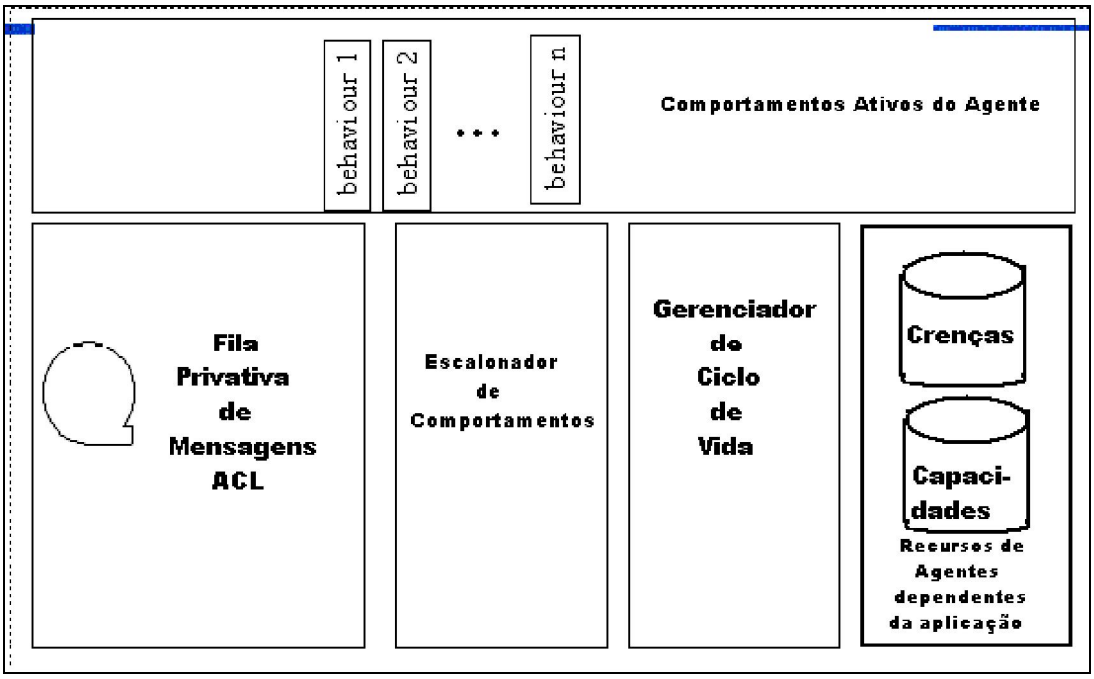

Figura 9. Arquitetura interna de um agente genérico em JADE
Fonte: Silva (2003)

Esses modos são citados da seguinte forma:

- AP_INIATED: o agente não tem nome nem endereço;
- AP_ACTIVE: o agente esta inscrito no AMS;
- AP_SUSPENDED: o objeto da classe esta no momento interrompido;
- AP_WAITING: o objeto esta no momento bloqueado, esperando algo;
- AP_DELETED: o agente está deletado ou encerrado;
- AP_TRANSIT. : o agente está migrando para algum destino;
- AP_COPY. : este estado é para o agente clonado; e
- AP_GONE: está em um estado estável.

Nessas especificações está descrito o ciclo de vida dos agentes em JADE, onde os mesmos têm alguns modos descritos pela FIPA (2003), como segue na Figura 10. Cada agente tem uma troca de estado tendo alguns termos que expressam o estado como:

- *public void doActivate*(): ativo ou espera;
- *public void doSuspend*(): suspenso;
- *public void doWait*(): faz passar do ativo para a espera;
- *public void doWake*(): passa do estado espera para o ativo;
- *public void doDelete*(): passa do estado ativo, espera ou suspenso para o deletado;
- *public void doMove* (Local destino): passa para o estado transito; e
- *public void doClone*: passa do estado ativo para o copy.

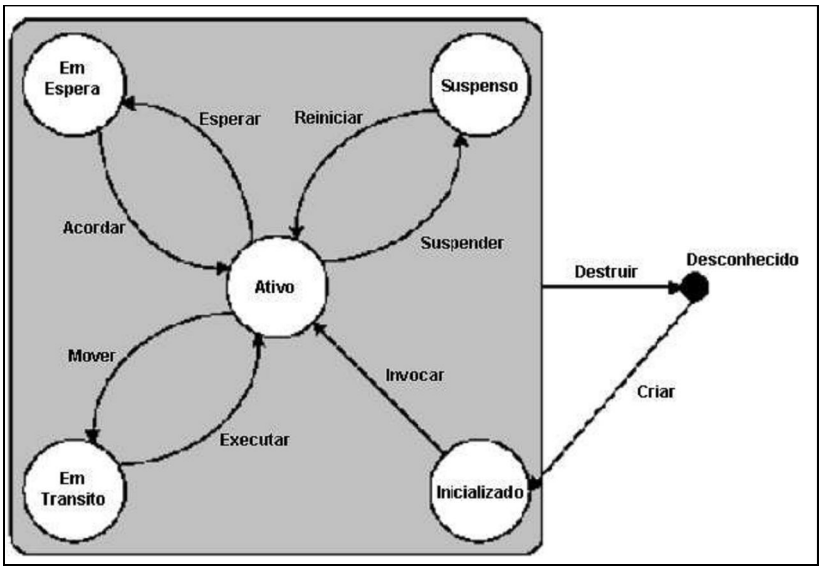

Figura 10. Ciclo de Vida de um agente definido pela FIPA
Fonte: Silva (2003)

Conforme Silva (2003), no entanto, é permitido ao agente executar seus comportamentos (*behaviours)* apenas quando estiver no estado ativo e se em qualquer dos comportamentos de um agente for chamado o método *doWait ()*, o agente como um todo e todas suas atividades serão bloqueadas, não só o comportamento que chamou o método. Ao invés disso, o método *block ()* da classe *Behaviour* evitaria essa bloqueio total, uma vez que permite a suspensão de apenas um comportamento, e além de se desbloquear caso uma mensagem for recebida.

No entanto, existem métodos importantes e particulares no *framework* de JADE, mas um dos principais métodos da classe *agent*, precisa ser esclarecido, principalmente a classe *public void addBehaviour*. Segundo Silva (2003), adiciona um novo comportamento ao agente. Ele será executado com outros comportamentos. Geralmente é usado no método *setup ()* para disparar algum comportamento inicial, porém pode ser usado também para gerar comportamentos dinamicamente. O método *removeBehaviour ()* faz justamente o inverso: remove determinado comportamento do agente.

2.4.6.4 Classe Behaviour

Conforme já explicado no tema acima o conceito de Behaviour, Silva (2003) explica ainda que é uma classe abstrata do JADE disponível no pacote *jade.core.behaviours.* Onde essa classe abstrata possui alguns métodos implementados e outros não, e não pode ser instanciada diretamente. Ela tem como finalidade provêr a estrutura de comportamentos para os agentes JADE serem programados. É no método *action*() que as ações serão desenvolvidas, sendo que o método *done*() informa o estado do comportamento, retornando *true* caso o comportamento tenha terminado e este é removido da fila de comportamentos, caso retorne *false*, indica que o comportamento não tenha terminado obrigando o método *action*() a ser executado novamente.

Além dos métodos citados acima, que são a parte principal do agente, existem outros métodos para auxiliar na manipulação dos comportamentos, como segue:

- *protected Agent myAgent :* Retorna o agente a qual esse comportamento pertence;
- *void block ():* Bloqueia esse comportamento. Pode ser passado como parâmetro um o tempo de bloqueio em milisegundos: *void block (long* tempo);
- *void restart ():* Reinicia um comportamento bloqueado;
- *void reset ():* Restaura o estado inicial do comportamento; e
- *boolean isRunnable ():* Retorna se o comportamento está bloqueado ou não.

No entanto, conforme Silva (2003), existem outras classes derivadas do Behaviour que alteram a forma como o comportamento do agente é executada, e que são listadas a seguir:

- *jade.core.behaviours.**CyclicBehaviour**:* Comportamento atômico que deve ser executado sempre. No caso, o método *done ()* herdado da super classe *Behaviour*,

sempre retorna *false*, o que faz com que o comportamento se repita como se estivesse em um “loop” infinito;

- *jade.core.behaviours.**TickerBehaviour**:* tarefas específicas.. No caso, o desenvolvedor redefine o método *onTick* () e inclui o trecho de código que deve ser executado periodicamente;
- *jade.core.behaviours.**WakerBehaviour**:* é executado depois de determinado tempo expirado. No caso, a tarefa é inserida no método *handleElapsedTimeout* ()é chamado sempre que o intervalo de tempo é transcorrido; e
- *jade.core.behaviours.**OneShotBehaviour**:* Comportamento atômico que é executado uma única vez. Essa classe abstrata pode ser herdada para a criação de comportamentos para tarefas que precisam ser executadas em apenas uma única vez. Tem como classe filha a classe *SenderBehaviour*. - *jade.core.behaviours.**SenderBehaviour*** – é um comportamento do tipo *OneShotBehaviour* para envio de mensagens ACL.

No desenvolvimento do projeto foi utilizado a linguagem JAVA, a qual é apresentada na próxima seção.

2.4.7 Linguagem JAVA

A linguagem JAVA foi desenvolvida em 1995 por Sun Microsystems, com intuito de criarem programas desenvolvidos em rede. Segundo Deitel (2000) o primeiro objetivo era ser utilizada em sistemas isolados com quantidade mínima de memória. Mas, com a explosão da internet e da World Wide Web, a linguagem Java gerou um interesse maior onde se percebeu um grande potencial da linguagem na criação de páginas web com conteúdo dinâmico. Conforme Deitel (2000), Java é composto basicamente por três elementos: uma linguagem orientada a objetos, um conjunto de bibliotecas e uma máquina virtual na qual os programas são interpretados.

Silva (2003), caracteriza o Java como uma linguagem simples, por seus autores se basearam na linguagem C++ para desenvolvê-la, mas retirando as características de orientação a objetos por serem raramente utilizadas. Também por causa de ser uma linguagem que utiliza o paradigma orientado a objeto para o desenvolvimento do software. Outra característica citada, diz respeito a variedade de classes para trabalho em redes que compõe o Java. Outra característica importante fala sobre a forma que são executados os programas, onde em vez de, o sistema operacional executar diretamente o programa, é utilizado um programa intermediário, chamado de *Java Virtual Machine*

(JVM), que por sua vez, interpreta o que foi desenvolvido pelo programador. Assim o mesmo programa pode ser executado em plataformas diferentes desde que haja uma JVM para a plataforma na qual se deseja executar o programa. Devido a natureza de execução do programa Java ser interpretada, a sua performance não é tão boa, mas há a possibilidade de se gerar o programa para uma plataforma em específico, fazendo com que a performance fique similar a programas desenvolvidos em C++. Cita Silva (2003), também a robustez da linguagem, onde são impostas restrições para evitar erros do programador, e assim minimizando os custos de desenvolvimento.

E em relação ao desenvolvimento de sistemas multiagentes, Bigus e Bigus (2001) relaciona algumas características importantes no desenvolvimento de Java:

- **Autonomia –** Aplicações em Java são processos separados e podem executar por um longo período de tempo. E, é possível implementar um agente como uma *thread* separada. A comunicação pode ser feita através de *sockets* ou invocação remota de métodos (*Remote Method Invocation* - RMI).
- **Mobilidade –** No Java essa característica é possível através da portabilidade do *bytecode* e dos arquivos JAR (executável do Java). Essa mobilidade também permite enviar os códigos pela rede e executar em outra máquina.

Pela sua mobilidade, pode atuar em sistemas distribuídos e em máquinas diferentes, utilizando tecnologias próprias para que opere nestes ambientes, sendo necessário o *Remote Method Invocation* (RMI) *que* é uma das ferramentas disponíveis que Java utiliza. Conforme Bigus e Bigus (2001), na chamada remota de métodos (RMI) que é um conjunto de classes e interface em Java que encapsulam muitos mecanismos de troca de dados, em alguma outra JVM (*Java Virtual Machine)*. Um objeto RMI é um objeto remoto em Java e seus métodos podem ser chamados de outra JVM, utilizado em rede.

3 DESENVOLVIMENTO

Neste capítulo serão descritos os processos para desenvolver o projeto, desde o seu planejamento, com levantamento de requisitos, regras de negócio, análise de riscos até a parte de implementação como, padrões utilizados, linguagem, tecnologia para implementação dos agentes (JADE) e sistemas gerados (Cliente e Servidor).

3.1 MODELAGEM

3.1.1 Requisitos do Sistema

Nesta seção são descritas a regras do jogo, como também identificados os requisitos funcionais e não funcionais do sistema.

3.1.1.1 Regras de Negócio:

O jogo possui uma série de regras que devem ser cumpridas, sendo elas:

- Somente o jogador que estiver criando uma partida pode definir quantos jogadores podem participar da mesma, e quantos serão humanos e quantos serão virtuais;
- Os jogadores não poderão ter exércitos com a mesma cor, ou seja, ao se conectar o sistema servidor gerencia as cores que ainda estão liberadas para uso;
- No início da partida devem ser distribuídas as cartas objetivo;
- Após a distribuição das cartas objetivo, devem ser distribuídas as cartas território, e distribuir os exércitos dos jogadores no mapa;
- A primeira rodada do jogo serve somente para os jogadores reporem os exércitos no mapa;
- A quantidade de exércitos que o jogador pode colocar no mapa a cada rodada, e na sua vez, é relativa à quantidade de territórios ocupados dividido por dois, sendo considerada apenas a parte inteira do resultado, ou seja, caso o jogador tenha 17 territórios em seu poder, o valor de sua metade é de 8.5, como não há uma metade de exército, a quantidade de exércitos a ser colocada é de 8;

- A quantidade mínima de exércitos que devem ser colocados a cada rodada, e na vez do jogador, é de três exércitos, ou seja, se o jogador tiver somente 4 ou cinco territórios, a quantidade de exércitos que ele deveria repor, conforme a regra, é de 2 exércitos, como o resultado é menor que a quantidade mínima, dessa forma o jogador deverá repor 3 exércitos;
- As disputas por território devem ser feitas com o mínimo de dois exércitos, para quem está atacando, e um exército para quem está defendendo o território. Quando o jogador ataca e vence a jogada, ele deve movimentar os exércitos atacantes para ocupar o território que acabou de ganhar, logo deve ficar no território de origem, o mínimo de um exército e, por isso, deve ser contada a quantidade de exércitos para ataque o total no território atacante menos um (o que fica no território atacante). Já quando se defende, há a possibilidade de perder o território, ou seja, todos os exércitos saírem do território, então conta-se com todos os exércitos do território para defesa;
- As disputas por território podem ser realizadas com o máximo de quatro exércitos por vez para quem está atacando, sendo usados somente três para o ataque, e três por vez para quem está defendendo o território;
- Um exército só poderá atacar territórios que fazem divisa com o território ocupado ou apontado por setas;
- As disputas se darão por intermédio de dados virtuais, sendo que serão utilizados dados de cor vermelha para representar o ataque, e dados amarelos para representar a defesa;
- Cada dado representa um exército, ou seja, se quem estiver defendendo tiver em seu território somente dois exércitos, serão utilizados para defesa somente dois dados, caso tenha mais de três exércitos serão utilizados três dados, o mesmo para quem ataca, sempre levando em conta que o exército atacante utiliza a quantidade que há no território menos um exército, sendo no máximo 3 exércitos de cada vez, o qual deve ficar no território;
- O resultado será apurado comparando os dados com maior resultado do exército que está atacando com o dado de maior resultado do exército que está sendo atacado, sendo que o maior entre eles será o vencedor. Em caso de empate a vitória é declarada para o exército de defesa. Em caso de confronto entre territórios onde há quantidades de exércitos inferiores a três, serão feitos os confrontos conforme a regra de quantidade de

dados, mas sendo descartados os dados cujo resultado tenha o menor valor até equivaler a quantidade de dados do outro jogador, não importando se está atacando ou defendendo;

- Em caso de vitória do exército atacante, o mesmo deve movimentar os exércitos usados para ataque ao território de conquista. O envio de exércitos a um território ocupado só conta com os exércitos que estão atacando, caso o jogador possua mais exércitos no território que está atacando, estes devem ficar no território até o momento oportuno para movimentação de exércitos;
- O jogador poderá atacar por tempo indefinido, parando somente se sobrar um exército em cada território ocupado por ele ou quando o jogador atacante assim o desejar;
- É permitido movimentar exércitos de um território ocupado para outro território ocupado, desde que faça divisa ou apontado por setas, e na quantidade máxima de três exércitos por território a cada rodada, sendo obrigatório ficar um exército no território simbolizando a ocupação do mesmo, ou seja, se o jogador movimentou três exércitos para um outro território, onde esse teria somente dois exércitos, ficando com cinco ao todo, o jogador poderá movimentar dois exércitos do território que acabou de receber reforço pois tinha somente essa quantidade antes de receber o reforço;
- Após a movimentação de exércitos, o jogador não poderá fazer mais nenhum ataque, até que os demais jogadores tenham jogado;
- O jogador pode adquirir uma carta território para troca mediante a conquista de um território durante a sua rodada, sendo que, se o jogador tentou atacar territórios, mas não teve êxito em nenhum ataque, fica impossibilitado de receber uma carta. Caso consiga adquirir pelo menos um território, indiferente da quantidade de derrotas tem o direito de receber a carta;
- A troca de carta por exércitos só será possível quando o jogador estiver iniciando a sua rodada, e no momento de repor exércitos no mapa, e tenha três cartas território com o mesmo símbolo, ou com os símbolos totalmente diferentes, ou ainda com duas cartas com o mesmo símbolo ou símbolos diferentes mais um coringa. Todas as cartas territórios possuem um símbolo, sendo que há somente três símbolos diferentes que podem estar impresso na carta, onde somente a carta coringa contém os três símbolos;

- As trocas de cartas são contadas sequencialmente, ou seja, um jogador fez uma troca, é contado como a primeira, o próximo faz outra troca, é contado como a segunda e assim sucessivamente;
- Ao obter cinco cartas, é obrigatório ao jogador realizar a troca de três dessas cinco cartas na próxima rodada, impossibilitando-o de adquirir novas cartas;
- Na troca de cartas por exércitos, caso o jogador, cujo exército, estiver ocupando uma das cartas território que estão sendo trocadas, tem por direito colocar um exército no território representado pela carta; e
- O jogador que alcançar o objetivo primeiro é considerado vencedor.

3.1.1.2 Requisitos Funcionais:

O sistema deve oferecer dois módulos, onde seja possível criar uma nova partida, sendo esse chamado de servidor, e outro onde o jogador possa conectar com o servidor e fazer parte de uma partida, sendo esse chamado de cliente.

No módulo servidor o sistema deve:

- Definir a quantidade de jogadores que farão parte do jogo, optando por jogadores humanos e virtuais;
- Definir a porta para conexão do servidor; e
- Definir o nome do servidor para que o cliente possa se conectar a ele.

No módulo cliente o sistema deve:

- Definir um nome para o jogador, que o representará na partida;
- Definir a cor de exército a ser distribuído no mapa;
- Definir o endereço IP do servidor;
- Definir o nome do servidor para conexão;
- Permitir o jogador distribuir os exércitos em qualquer território que esteja ocupado por ele;
- Permitir selecionar dois territórios ao mesmo tempo, um de sua posse e outro inimigo, a fim de atacá-lo;

- Ao fim de suas jogadas, permitir a movimentação de exércitos entre territórios de sua ocupação e que fazem divisa;
- Permitir o jogador adquirir uma carta território ao final de sua jogada de ataque para possível troca, caso tenha ganhado algum território durantes os ataques; e
- Permitir a troca de cartas recebidas, de três em três, cujo símbolo, seja igual nas três cartas ou totalmente diferentes.

3.1.1.3 Requisitos não Funcionais:

Os requisitos não funcionais determinam quais restrições o projeto terá, como também

- O sistema deve ser composto por dois módulos, um servidor e outro cliente;
- O sistema deve ser desenvolvido em linguagem Java;
- O desenvolvimento do sistema deve utilizar uma interface de desenvolvimento livre de cobrança de licença;
- O módulo cliente deve se conectar somente com o servidor; e
- A implementação dos agentes inteligentes utiliza o *framework* JADE;

3.1.2 Casos de Uso

Para modelagem do sistema, foi utilizada a UML. O diagrama de caso de uso (Figura 11) mostra quais ações o jogador poderá realizar no sistema.

Cada um dos realizam funções distintas e que se complementam, e que por isso possuem opções diferentes. O módulo servidor oferece as seguintes funções:

Criar Servidor: cria um novo jogo, onde define quantos jogadores poderão fazer parte da partida, verifica se todos já estão conectados e prontos e defini a quantidade de jogadores virtuais que participarão do jogo.

Parâmetros de Jogo: o jogador define todos os mesmo parâmetros do modo cliente, e também a quantidade de jogadores que podem fazer parte do jogo, quantos serão jogadores virtuais e que a partida deve iniciar.

O módulo cliente possui as seguintes funções:

Conectar: possibilidade do jogador participar de uma partida em início, onde define o nome do mesmo no jogo, qual sua cor de exército e que está pronto para iniciar a partida.

Parâmetros de Jogo: em modo cliente, o jogador define o nome, cor de exército e se está pronto para o início da partida. Em modo servidor, o jogador define todos os mesmo parâmetros do modo cliente, e também a quantidade de jogadores que podem fazer parte do jogo, quantos serão jogadores virtuais e que a partida deve iniciar.

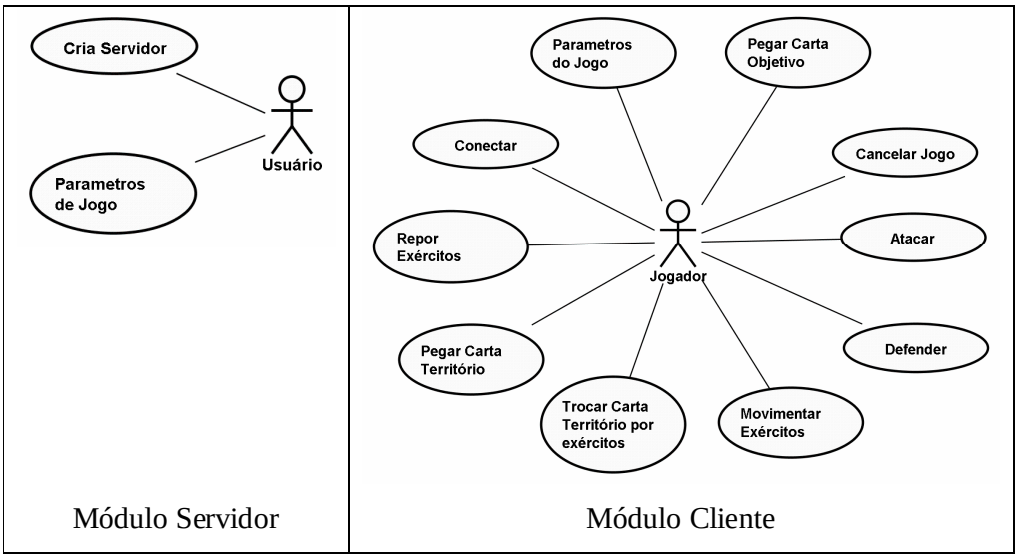

Figura 11. Diagrama de *USE CASE*

Pegar Carta Objetivo: o jogador no início do jogo deve pegar uma carta objetivo, sendo jogador humano ou virtual, o qual o jogador deve cumprir para poder vencer a partida.

Repor Exércitos: no início da jogada o jogador coloca mais exércitos no mapa, reforçando o seu exército e possibilitando ataques.

Cancelar Jogo: somente no servidor, o jogador pode cancelar uma partida.

Atacar: O jogador seleciona o território do qual vai atacar, e depois seleciona o território a ser atacado. Logo após o atacante seleciona a opção de atacar onde o sistema sorteia os valores de ataque e defesa conforme a quantidade de exércitos de cada território. O sistema compara os resultados de ataque e defesa avaliando o dado com maior valor de ataque com o dado de maior valor de defesa, sendo que ganha aquele que obtiver o maior valor, em casos de empate a vitória é declarada para o exército defensor.

Movimentar Exércitos: O jogador que estiver atacando, indica que não vai mais continuar a sua jogada e movimenta os exércitos de um território a outro, desde que elas façam fronteiras, sendo que a quantidade de exércitos a serem movimentados não deve ultrapassar 3 exércitos por rodada.

Pegar Carta Territóio: Após terminar a jogada e movimentar os exércitos, o jogador tem direito a pegar uma carta, caso tenha ganhado novos territórios durante sua jogada, retirando ela do monte de cartas território e coringa misturados.

Na Figura 12 são apresentadas as classes que compõe o projeto.

Classe Continente: Conterá o nome do continente e a quantidade de exércitos a ser recebido no momento de repor exércitos caso o jogador obtenha a totalidade dos territórios que faz parte.

Classe Território: Agrega a classe Continente, e representa os territórios do mapa, contendo o nome do território que representa, o exército ocupante, quantidade de exércitos que estão ocupando o território e uma lista dos territórios que fazem divisa com o mesmo.

Classe Carta Território: Representa um território com um símbolo.

Classe Símbolo: São os símbolos que combinados em uma carta território possibilitam uma troca por exércitos ao jogador.

Classe Carta Coringa: Contém todos os símbolos e pode ser combinada com outras duas cartas para proporcionar uma troca de exércitos para o jogador.

Classe Carta Objetivo: Contém o código do objetivo, o qual deve ser cumprido para que o jogador seja vencedor.

Classe Jogador: Contém nome, exército escolhido, IP do computador e se o jogador é humano ou virtual, representando assim o jogador.

Classe Gerencia Conexão: Contêm o IP, código da mensagem, mensagem, status do envio da mensagem e um status do recebimento, servindo para fazer a ligação cliente-servidor e servidor-cliente.

Classe Exército: Contêm a cor do exército.

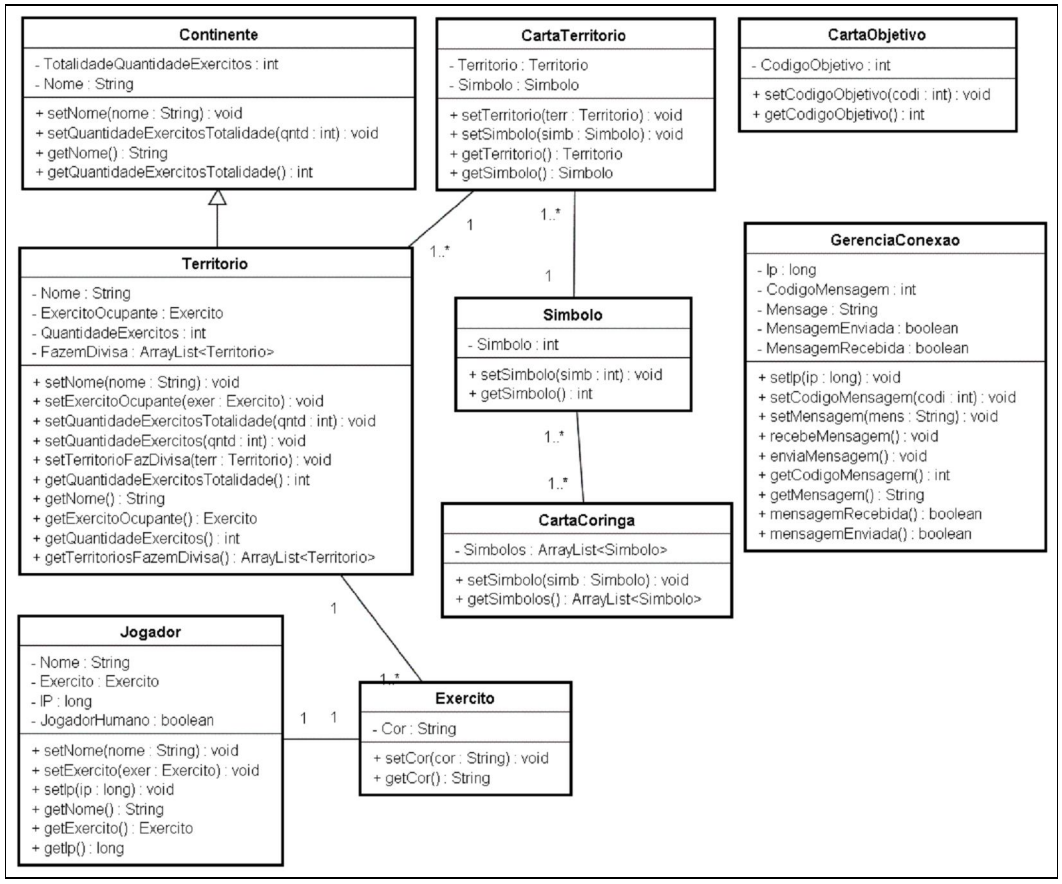

Figura 12. Diagrama de Classes

Demais diagramas que expliquem melhor o funcionamento do sistema, como os diagramas de atividades, de estado e de seqüência estão descritos nos apêndices A.1, A.2 e A.3 respectivamente.

3.1.3 Árvore de Decisão

A inteligência dos agentes é representada através de árvores de decisão. Essas árvores têm uma estrutura inicial formada pelo objetivo, em seguida pela ação que deve ser tomada, sendo elas, repor exércitos no mapa, atacar ou movimentar tropas e em seguida as regras para efetuar tal ação (Figura 13).

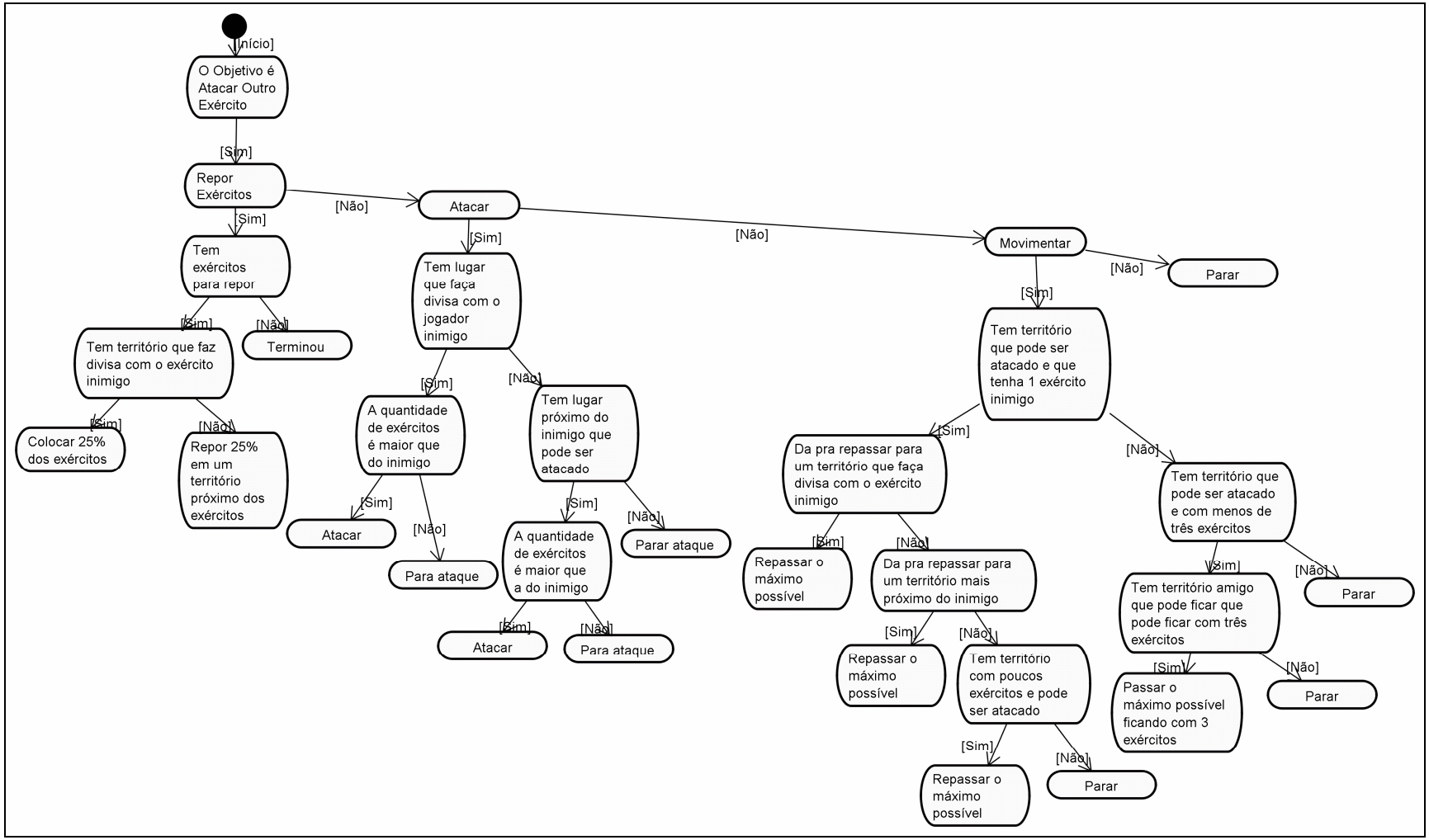

Figura 13. Árvore de decisão para atacar um outro exército

A Figura 14 demonstra como é o funcionamento da árvore. O primeiro nível avalia o objetivo, no caso, atacar outro exército. O seguinte verifica qual o tipo de ação a ser executada, que pode ser repor exércitos, atacar ou movimentar. Os níveis seguintes são as regras que determinam quais ações o agente tomará no jogo.

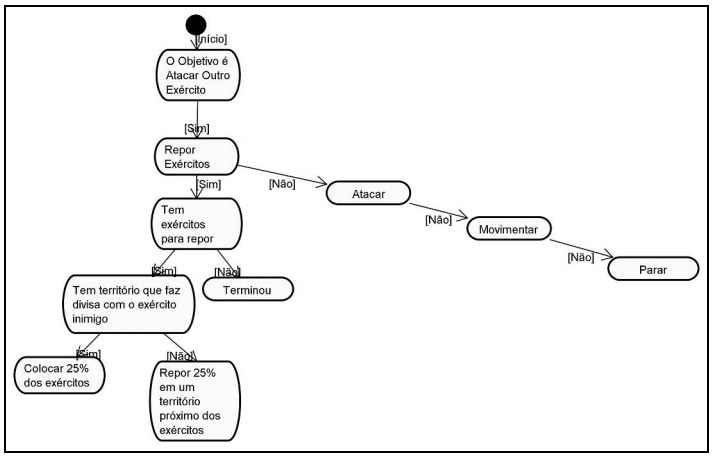

Figura 14. Árvore de Decisão – Atacar Outro Exército – Reposição

A Figura 15 apresenta a segunda parte da árvore de decisão do objetivo de Atacar Outro Exército. Essa parte mostra as regras para que seja efetuado um ataque.

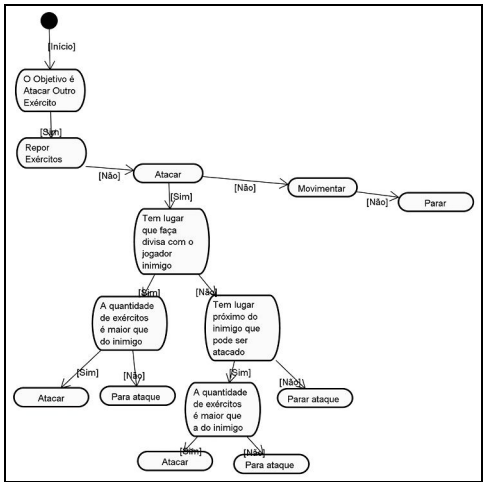

Figura 15. Árvore de Decisão – Atacar Outro Exército – Atacar

A Figura 16 apresenta a terceira parte da árvore de decisão do objetivo de Atacar Outro Exército. Essa parte mostra as regras para a movimentação de exércitos no final da rodada.

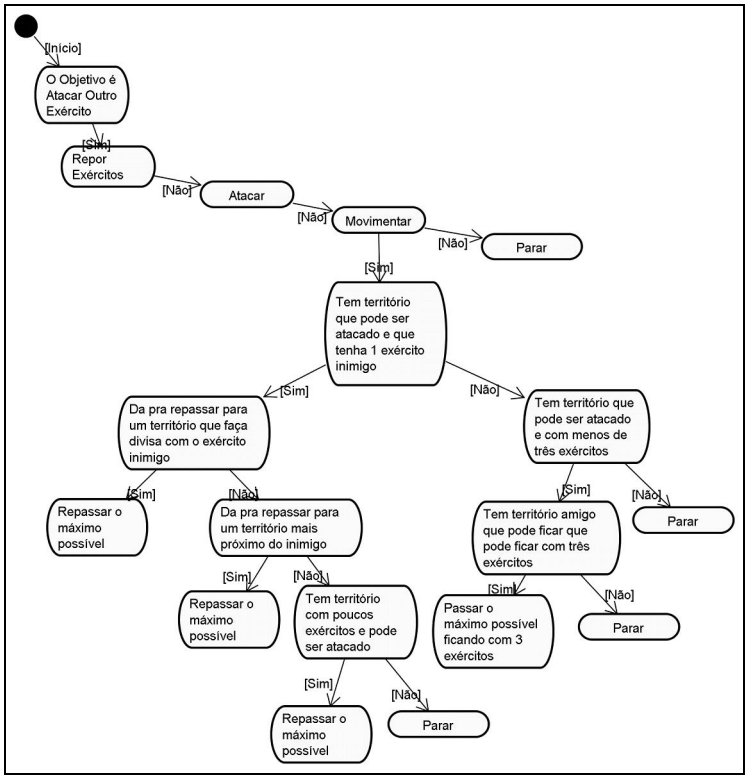

Figura 16. Árvore de Decisão – Atacar Outro Exército – Movimentar

Foi criada uma árvore genérica (Apêndice A.4.1), que além de ser utilizada para testar o funcionamento do agente inteligente, também é utilizada para os objetivos que não possuem uma árvore de decisão. Até o momento só foi desenvolvida essa árvore genérica e a árvore para o objetivo de atacar outro exército.

3.1.4 Análise de Riscos

A opção de o jogo rodar em diversos computadores é um risco a ser considerado, pois o servidor poderá demorar a responder por algum motivo no meio da partida, ou mesmo, os clientes ficarem fora de sincronismo durante o jogo. Para resolver esse problema a classe que gerencia a conexão ficará mantendo um diálogo com o servidor, verificando se o mesmo está no ar. Caso o servidor não responda durante certa quantidade de tempo será cancelada a conexão e finalizado o jogo. Além de manter uma lista de atualizações para cada cliente em separado, sendo essa lista anexada a classe jogador, então o cliente terá uma classe jogador e o servidor terá a mesma classe, sendo que as alterações realizadas na classe jogador do servidor serão repassadas para a classe jogador do cliente, e assim atualizando o jogo para o cliente.

Com a resolução do risco de se perder a conexão, é criado um novo risco da partida ser interrompida inesperadamente, para amenizar esse problema, pode ser criado um mecanismo que salve o jogo em um certo período de tempo, e possibilitando a continuação de um jogo salvo.

Os jogadores virtuais têm a mesma base de conhecimento, logo podendo jogar da mesma forma, inicialmente pelo jogo ser voltado ao objetivo, cada jogador tende a atacar e defender de forma diferenciada, mas caso haja uma tendência de os jogadores virtuais jogarem da mesma forma, há a possibilidade de incluir uma personalidade a cada um, sendo ela defensiva, agressiva ou normal. Ainda há a possibilidade de incluir objetivos secundários como, por exemplo, atacar outro jogador somente para enfraquecê-lo, já que o próprio objetivo pareça impossível de alcançar.

3.2 Implementação

A implementação do jogo foi dividida em dois sistemas, sendo um o sistema servidor, que gerencia parte do jogo provendo métodos compartilhados para o cliente, e o sistema cliente, que deve gerenciar a interação do jogador e o jogo, liberando as ações e atualizando o mapa conforme o passar do jogo. Os jogadores virtuais são implementados utilizando-se o JADE (*Java Agent Development Framework*), um *framework* para auxiliar o desenvolvimento de agentes voltado para a linguagem Java, e que implementa padrões definidos pela FIPA *(*Foundation for Intelligent Physical Agents*)*.

3.2.1 Implementação dos Agentes

A implementação dos agentes foi feita com a linguagem Java e com o auxilio do *framework* JADE. Para a criação de um agente com o uso do JADE, deve ser fornecido o tipo de comportamento que o agente terá. Para o projeto foi cogitado a utilização do *CyclicBehaviour*, *OneShotBehaviour* e *SimpleBehaviour*. A escolha foi feita pelo *SimpleBehaviour* pois há o controle direto da forma que o comportamento reage a partir do uso do método *action*, onde é colocado o código que servirá de comportamento para o agente, e do método *done* que informa quando o comportamento deve parar. A Figura 17 contém a classe Java com o código que é utilizado para gerar o comportamento do agente, onde herda do JADE a classe SimpleBehaviour e assim sendo necessário implementar os métodos *action e done*.

```
/** Declara uma classe Java herdando do JADE o SimpleBehaviour
public class ComportamentoPadrao extends SimpleBehaviour{

    /** Indica que o agente terminou */
    private boolean terminou = false;

    /** Determina qual a ação atual do jogador virtual */
    private Enumerados.acoes acaoAtual = Enumerados.acoes.inicio_jogada;

    /** Creates a new instance of ComportamentoPadrao */
    public ComportamentoPadrao() {
    }

    public void action() {

        if (acaoAtual == Enumerados.acoes.inicio_jogada){
            inicioJogada();
            // acao de distribuir exércitos
        }else if (acaoAtual == Enumerados.acoes.ataque){
            atacarTerritorio();
            // acao de atacar
        }else if (acaoAtual == Enumerados.acoes.redistribuir){
            redistribuirExercitos();
            // acao de redistribuir os exércitos no mapa
        }
    }

    public boolean done() {
        return getTerminou();
    }

    /** Retorna se o comportamento terminou ou não
     *
     *@return true se terminou e false se não terminou
     */
    public boolean getTerminou(){ return terminou;}

    /** Seta para o comportamente se terminou ou não
     *
     *@param ter terminou (true) ou não terminou (false)
     */
    public void setTerminou(boolean ter){terminou = ter;}
}
```

Figura 17. Código para o comportamento padrão do agente

Conforme a variável “acaoAtual” será executada uma ação ao qual fará a consulta na base de conhecimento do agente. Para a criação do agente são utilizadas classes do JADE que controlam o funcionamento do mesmo. Para o funcionamento é necessário utilizar a classe Runtime, ProfileImpl, que fornecem o local para criação do agente, chamado de container. Com essas duas classes instaciadas é possível criar um ContainerController que serve para criar e agrupar os agentes conforme mostra a Figura 18. Em controller.createNewAgente, é efetivada a criação do agente, onde é passado em seus parâmetros o nome do agente dentro do JADE, a classe que serve como base e os argumentos para criação, em seguida o agente é iniciado e ativado, dessa forma entrando em execução.

```
Runtime runtime = Runtime.instance();
Profile profile = new ProfileImpl();

// Cria o agente dentro do container de agentes
ContainerController container_controller = runtime.createMainContainer(profile);
AgentController agent_controller =
container_controller.createNewAgent("jogador”,"br.com.tcc.war.Agente.Componentes.JogadorVirtual",
        new Object[]{new Objetct()});

agent_controller.start();
agent_controller.activate();
```

Figura 18. Código para criar e iniciar um agente JADE

Para a implementação, foram criadas duas classes (Figura 19), onde a classe Agente estende funcionalidades da classe Agent do *framework* , e também a classe ComportamentoPadrão que estende as funcionalidades da classe *SimpleBehaviour*. O sistema trabalha com somente um objeto da classe Agente, não importando quantos jogadores virtuais foram adicionados ao jogo. Como cada jogador executa suas ações no jogo um de cada vez, não há a necessidade de ter vários agentes, um para cada jogador virtual, bastando que ao consultar a base de conhecimento, seja definida a sua árvore de decisão conforme o seu objetivo. Para isso foi criado um método no comportamento padrão, chamado setArvoreDecisao, que é atribuído toda a vez que se troca o jogador que está jogando. Dessa forma um único agente gerencia todos os jogadores virtuais do jogo.

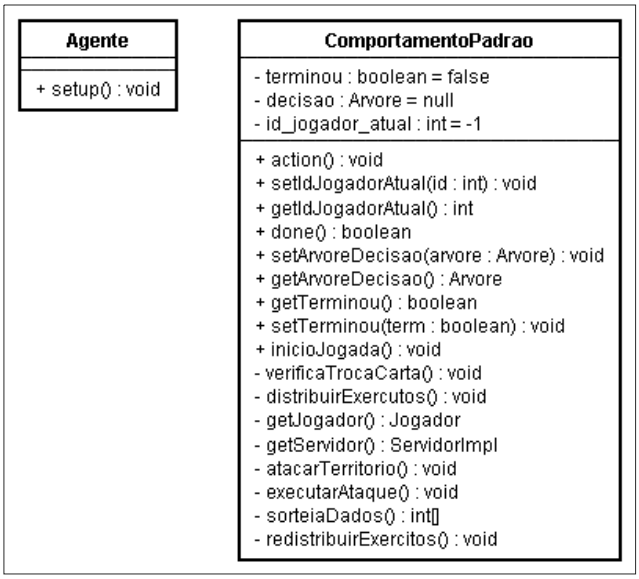
Agente
+ setup() : void
ComportamentoPadrao
- terminou : boolean = false
- decisao : Arvore = null
- id_jogador_atual : int = -1
+ action() : void
+ setIdJogadorAtual(id : int) : void
+ getIdJogadorAtual() : int
+ done() : boolean
+ setArvoreDecisao(arvore : Arvore) : void
+ getArvoreDecisao() : Arvore
+ getTerminou() : boolean
+ setTerminou(term : boolean) : void
+ inicioJogada() : void
- verificaTrocaCarta() : void
- distribuirExercutos() : void
- getJogador() : Jogador
- getServidor() : ServidorImpl
- atacarTerritorio() : void
- executarAtaque() : void
- sorteiaDados() : int[]
- redistribuirExercitos() : void

Figura 19. Classes para instanciar o agente no jogo

3.2.2 Implementação das Regras

A implementação das regras da árvore de conhecimento, utilizam um ferramenta chamada JCLIPS que permite a inclusão de regras e fazer a consulta na base de conhecimento.

3.2.3 Usando o JCLIPS

Para interagir o sistema com o JCLIPS é necessário utilizar uma biblioteca de classes chamada JCLIPS.JAR. Inicialmente deve ser criada um classe implementando a interface "Observer", a qual possui um método chamado "update" o qual recebe as informações geradas pelo JCLIPS.

Um objeto instaciado da classe JClips deve ser utilizado para configurar e inicializar a API. É nele que é informado o arquivo com as regras da árvore de decisão, como também qual a regra que inicia os testes de conhecimento (Figura 20).

```
Public class JClipsExample implements Observer {
    private String base_conhecimento = "myknowledge.clp";
    public void update(Observable jClips, Object messageObj) {
        String message = (String) messageObj;
        System.out.println("Message received: " + message);
    }

    public static void main(String[] args) {
        System.out.println(JClips.getVersion());
        JClips jClips = JClips.getInstance();
        jClips.init();
        jClips.addObserver(new JClipsExample());
        try {
            URL fileUrl = JClipsExample.class.getResource(base_conhecimento);
            if (fileUrl == null) {
                throw new FileNotFoundException(base_conhecimento);
            }
            File file = new File(new URI(fileUrl.toString()));
            jClips.load(file.getPath());
            jClips.reset();
            jClips.run();
            jClips.executeCommand("(assert (ola_mundo))");
            jClips.run();
        } catch (Exception e) {e.printStackTrace(System.err);}
        jClips.deallocate();
    }
}
```

Figura 20. Código de Exemplo de Conexão com o JCLIPS

Ao gerar um arquivo de regras é necessário passar as informações para as regras como parâmetros na string que executa a regra (jClips.executeCommand("(assert (ola_mundo))") (Figura 20).

As regras são incluídas em JCLIPS a partir de um arquivo texto, onde para isso há uma estrutura previamente definida (Figura 21).

```
(defrule assert-result
    (initial-fact)
  =>
    (printout t "Firing rule assert-result" crlf)
    (assert (result "The answer to life, the universe and everything!"))
)
```

Figura 21. Regra em JCLIPS

Como são incluídas em um arquivo texto, elas ficam fixas, tanto as regras quanto os valores que devem ser avaliados nas regras. A resolução para esse problema seria gerar estatísticas no sistema e associa-lás a marcações, como por exemplo “<@QNTDEXER>” onde “QNTDEXER” é o nome do marcador, que antes de fazer a consulta na base de conhecimentos, o sistema pegaria o arquivo texto contendo a regra a ser avaliada, verificando se acha algum marcador a ser substituído, ao final do processo gerando um novo arquivo texto com os valores atualizados e prontos para serem avaliados. Dessa forma a inteligência poderia ser aperfeiçoada criando novas estatísticas no sistema e novas marcações, e também alterando o arquivo texto contendo a inteligência a ser modificada, e assim aperfeiçoando as regras do agente.

3.2.3.1 Problemas no Uso do JCLIPS

Após ser testado as regras de exemplo que está no site do JCLIPS, não foi encontrado problema em sua execução. Ao se tentar executar uma regra para cálculo de média (Figura 22), o JCLIPS retorna um erro de sintaxe na definição da regra, sendo que esse erro acontece no momento em que é carregado o arquivo de regras. Conforme consulta feita na documentação do CLIPS, a sintaxe está correta, e não foi achado uma alternativa para poder se contornar o erro. Devido a esse problema, optou-se por criar uma classe com as regras aninhadas em testes lógicos, para poder fazer o agente interagir com o jogo.

```
(defrule send-result
    (result ?result)
  =>
    (printout t "Firing rule send-result" crlf)
    (send-to-java ?result)
)

(defrule primeiraEtapa
    (verificarMedia ?l1 ?l2 ?13)
    (assert media (/ (+ (+ ?l1 ?l2) ?l3) 3))
    (test (>= media 7))
  =>
     (assert (result "Media Aprovada"))
)
```

Figura 22. Regras para calcular de média

3.2.3.2 Solução

Para implementar as regras da inteligência do agente foi criado uma interface que serve como base para a implementação de novas árvores de decisão, que é a interface Arvore (Figura 23 – Interface). A utilização dessa interface não descarta o uso de JCLIPS, mas serve para definir uma forma única de acesso as consultas na base de conhecimento. Isso é possível pela interface conter métodos que devem ser implementados pela classe que representa a árvore a ser escrita, e que são acionados pelo sistema no momento em que deve ser tomada uma decisão pelo agente.

A classe ArvoreGeral (Figura 23 – Classe para árvore geral), implementa os métodos da interface e mais alguns métodos específicos. Então o sistema aciona os métodos da interface, que por sua vez acionam os métodos internos da árvore, executando a consulta nas regras e retornando os resultados para o agente, permitindo que possa executar as suas ações no jogo.

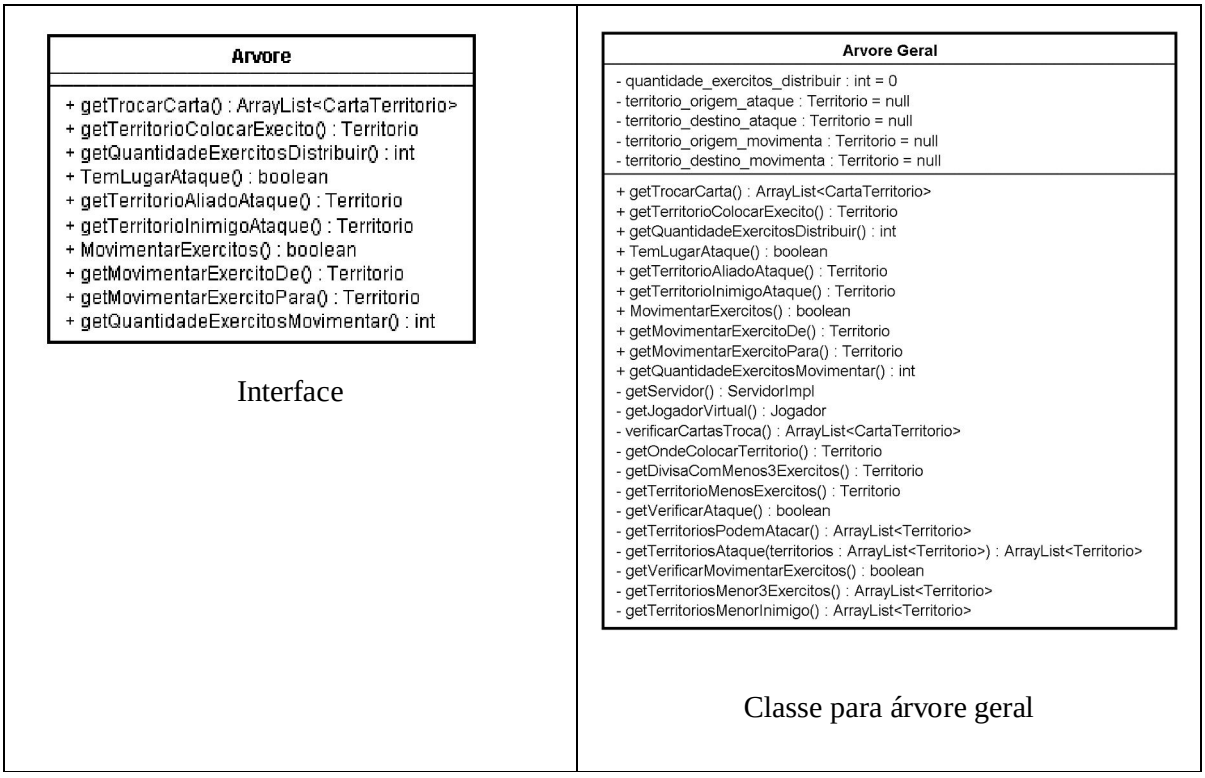
Arvore
+ getTrocarCarta() : ArrayList<CartaTerritorio>
+ getTerritorioColocarExecito() : Territorio
+ getQuantidadeExercitosDistribuir() : int
+ TemLugarAtaque() : boolean
+ getTerritorioAliadoAtaque() : Territorio
+ getTerritorioInimigoAtaque() : Territorio
+ MovimentarExercitos() : boolean
+ getMovimentarExercitoDe() : Territorio
+ getMovimentarExercitoPara() : Territorio
+ getQuantidadeExercitosMovimentar() : int
Interface
Arvore Geral
- quantidade_exercitos_distribuir : int = 0
- territorio_origem_ataque : Territorio = null
- territorio_destino_ataque : Territorio = null
- territorio_origem_movimenta : Territorio = null
- territorio_destino_movimenta : Territorio = null
+ getTrocarCarta() : ArrayList<CartaTerritorio>
+ getTerritorioColocarExecito() : Territorio
+ getQuantidadeExercitosDistribuir() : int
+ TemLugarAtaque() : boolean
+ getTerritorioAliadoAtaque() : Territorio
+ getTerritorioInimigoAtaque() : Territorio
+ MovimentarExercitos() : boolean
+ getMovimentarExercitoDe() : Territorio
+ getMovimentarExercitoPara() : Territorio
+ getQuantidadeExercitosMovimentar() : int
- getServidor() : ServidorImpl
- getJogadorVirtual() : Jogador
- verificarCartasTroca() : ArrayList<CartaTerritorio>
- getOndeColocarTerritorio() : Territorio
- getDivisaComMenos3Exercitos() : Territorio
- getTerritorioMenosExercitos() : Territorio
- getVerificarAtaque() : boolean
- getTerritoriosPodemAtacar() : ArrayList<Territorio>
- getTerritoriosAtaque(territorios : ArrayList<Territorio>) : ArrayList<Territorio>
- getVerificarMovimentarExercitos() : boolean
- getTerritoriosMenor3Exercitos() : ArrayList<Territorio>
- getTerritoriosMenorInimigo() : ArrayList<Territorio>
Classe para árvore geral

Figura 23. Interface e Classe para árvore de decisão

Com essa solução é possível continuar com o desenvolvimento do sistema, e permite que as regras sejam implementadas, não somente na linguagem em que foi desenvolvido o sistema, mas como também com ferramentas de terceiros.

3.2.4 Comunicação entre Sistemas

A técnica de comunicação define como o sistema realiza a comunicação entre o cliente e o servidor. Inicialmente seriam utilizadas funções da linguagem Java, que se utiliza de *socket* para envio e recebimento de mensagens. Logo essas mensagens seriam interpretadas e traduzidas em ações que o sistema deveria realizar. Esse sistema funciona muito bem, em uma rede local ou mesmo pela internet.

Já o RMI, permite ao cliente acessar métodos compartilhados pelo servidor, logo não é mais necessário interpretar mensagens para realizar ações, sendo que o cliente pode executar a ação desejada diretamente no servidor, da mesma forma que faz em uma outra classe Java qualquer. O RMI utiliza a mesma estrutura de comunicação que a técnica de *socket*, ou seja, conecta-se ao servidor por intermédio de um endereço IP e uma porta, proporcionando da mesma forma conectar a qualquer computador que tenha o protocolo TCP/IP, sendo ele em rede local ou pela internet. Sendo assim optou-se por utilizar RMI para a comunicação entre os dois sistemas.

Para realizar essa comunicação no Java, implementadas no servidor, deve haver basicamente duas classes, uma declarada como *interface* estendendo da classe *Remote*, o qual define quais métodos serão compartilhados com o cliente, sendo essa classe chamada Servidor.java, no sistema. Outra classe que contém a implementação desses métodos, que deve ser definida como *implements* dessa classe, sendo ela a classe ServidorImpl.java. Ao compilar o sistema, o Java gera um arquivo com o nome da classe que implementou os métodos incluindo no nome o sufixo "_stub.class", esse arquivo deve ser incluído ao projeto do cliente juntamente com a classe que serve de interface e contém o cabeçalho dos métodos compartilhados.

Com o RMI, além da possibilidade de executar um método remoto, há a possibilidade de transferir objetos entre os sistemas, sendo que a classe desses objetos deve conter em sua declaração "*implements Serializable*", que permite o envio do objeto utilizando essa conexão.

3.2.5 Servidor

É um sistema que gerencia o jogo, e para isso prove métodos que são executados pelo cliente remotamente via RMI. Somente o servidor disponibiliza métodos para acesso, sendo a transmissão de informações é feita pela passagem de parâmetros aos métodos e recebendo os seus retornos.

Além do RMI, a linguagem dispõe de característico para organizar o código fonte gerado, sendo um deles o pacote. É possível criar áreas para armazenar certos tipos de código fonte, e dessa forma podendo ser classificados por tipos de códigos. No caso do jogo foram classificados conforme a função que realizam no sistema. Um padrão comumente utilizado por desenvolvedores Java, os pacotes devem vir com o domínio da internet para facilitar a integração de pacotes de terceiros, sendo assim os pacotes relacionados com o jogo e pertencentes ao sistema servidor começam com “br.com.tcc.war.server”. Então no sistema servidor temos os pacotes iniciando em “br.com.tcc.war.server”, continuando a nomenclatura como segue:

- .base: neste pacote estão contidos classes que formam a base do sistema, como classes que modelam os territórios, continentes, exércitos, jogadores e etc;
- .controle: contém classes que controlam o funcionamento do sistema, são classes que contém as regras de negócio;
- .gui: as classes deste pacote são geralmente telas que devem ser usadas para interação do usuário com o sistema;
- .principal: nestes pacotes estão classes que fazem parte da tela principal do sistema; e
- .servidor: as classes que fazem parte deste pacote são aquelas que disponibilizam métodos a serem acessados por outros sistemas via RMI.

O servidor controla a lista de cartas-território que devem ser distribuídas aos jogadores. A cada fim de jogada, o jogador que conseguiu obter um ou mais territórios tem direito a uma carta território, a qual poderá ser utilizada mais à frente para obter mais exércitos. O servidor detém a lista das cartas como também a ordem em que serão entregues.

As ações que o jogador pode realizar são liberadas a partir do servidor. O cliente consulta o servidor para saber se é a vez do seu jogador jogar, e qual a ação que pode ser liberada a ele. No decorrer da partida, conforme as jogadas, ele vai definindo de quem é a vez de jogar, e o que esse jogador pode fazer, em contrapartida o cliente libera as opções ao jogador, para ele poder realizar as suas jogadas.

O servidor mantém o mapa, desde a distribuição dos territórios, como também as fronteiras, continentes e exércitos que estão ocupando cada território. Basicamente o servidor mantém quase todas as operações e informações do jogo, ficando a cargo do cliente mostrar os resultados ao

jogador como também interagir no jogo. Ficou a cargo do servidor manter os agentes que estão funcionando no momento e gerenciá-los.

3.2.5.1 Telas do Programa Servidor

Ao se iniciar o servidor é apresentada a tela principal do programa Servidor (Figura 24), essa tela disponibiliza o menu Jogo, ao qual é possível criar um jogo, parar um jogo em execução ou sair do sistema.

Figura 24. Tela principal do Servidor

Ao acessar o menu Jogo, opção Cria Servidor, é apresentada uma tela com as configurações do servidor, como é mostrado na Figura 25.

Figura 25. Tela Cria Servidor

Os campos e suas funções são relacionadas a seguir:

- Porta: Nesse campo deve ser informada a porta ao qual será utilizada para conexão dos clientes ao servidor;

- Nome Servidor: O nome do servidor serve para definir o nome do serviço que estará funcionado na porta informada;
- Jogadores: Define a quantidade de jogadores humanos farão parte do jogo; e
- Virtuais: Define quantos jogadores virtuais devem fazer parte do jogo.

Após as configurações serem informadas ao servidor e confirmadas, caso fora indicado jogadores humanos para a partida, é apresentada uma tela (Figura 26) onde é aguardado os jogadores humanos se conectarem.

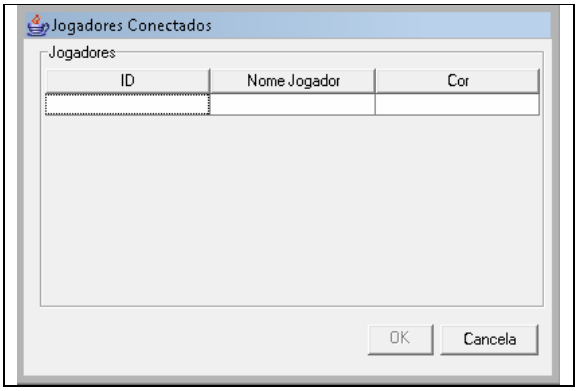

Figura 26. Aguardando conexão de outros jogadores

Após os clientes estarem conectados ao jogo, a tela é fechada e um processo gerenciador é iniciado e mantém o jogo em funcionamento.

3.2.6 Cliente

O cliente é um sistema que serve para fazer a interface entre o jogador e o jogo. Para tanto, o cliente se conecta ao servidor a fim de consultar informações e apresentá-las na tela e também para liberar as ações que o mesmo pode realizar em um determinado momento. Ao conectar-se o cliente inicia uma *thread* que fica consultando o servidor para saber que ações ele pode tomar, sendo que pode ser uma atualização de tela, ou mesmo a vez do jogador jogar.

Como o servidor, o cliente segue as mesma regras de pacotes e nomenclaturas, dessa forma os pacotes pertencentes ao cliente começam com “br.com.tcc.war.client”. Então no sistema cliente terão os pacotes iniciando em “br.com.tcc.war”, continuando a nomenclatura como segue:

- .client.controle: neste pacote estão as classes que realizam operações que intermediam a tela com outras classes. São elas que executam a maioria das regras de negócio;
- .client.gui: são classes que disponibilizam telas para o usuário;
- .client.servico: este pacote foi criado especificamente para manipular a conexão com o servidor;
- .server.base: este pacote é originado do sistema servidor, cuja as classes devem ser implementadas somente no servidor e não podem ser alteradas no cliente, são as classes que formam a base do jogo, como classes que modelam os territórios, continentes, cartas entre outras;
- .server.servidor: este pacote também é originado do servidor, o qual tem as classes que com os métodos remotos disponibilizados no servidor e que podem ser acessados pelo cliente; e
- componentes: este pacote é diferente dos outros pois não segue a mesma regra de uso do domínio de internet, nesse pacote estão classes que estenderam suas funcionalidades de algum componente do Java e que foram customizadas para melhor se adequarem ao sistema.

A organização dos pacotes foi montada para permitir uma melhor manutenção do sistema, visando uma programação em camadas.

3.2.6.1 Telas do Programa Cliente

O cliente em sua tela principal, disponibiliza o menu jogo, que permite ao jogador conectar-se a uma partida, a partir da opção conectar, ao qual é possível passar as configurações realizadas no servidor e assim conectar-se a ele, e sair para poder finalizar o programa, conforme Figura 27.

Figura 27. Tela principal do programa cliente

Ao tentar conectar o cliente ao servidor é apresentada uma tela de configuração (Figura 28), devem ser passados os dados do servidor e do jogador para que ele possa entrar em uma partida.

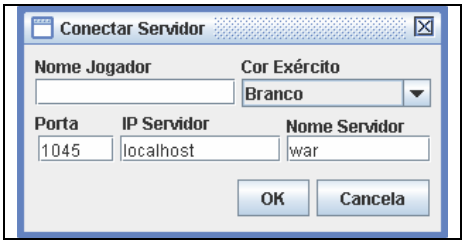

Figura 28. Conectar com o Servidor

Os campos e suas funções são listados a seguir:

- Nome Jogador: É informado o nome do jogador que será apresentado em uma lista com o nome dos participantes no mapa do jogo;
- Cor exército: Cada jogador deve selecionar uma cor, ao qual será utilizada para identificar quais territórios são de posse do jogador, e quais são dos adversários;
- Porta: É o número da porta que o servidor está executando;
- IP Servidor: Número IP da máquina em que o servidor está rodando; e
- Nome Servidor: Nome determinado para o servidor ao se criar um jogo.

Esses dados devem ser informados conforme as configurações definidas no servidor para que os programas possam estabelecer uma conexão e assim poder se comunicar. Após o servidor aceitar a conexão do cliente e informar que todos os jogadores estão conectados, então é possível continuar com partida.

Figura 29. Mapa do Jogo

A Figura 29 possui vários quadros, cada um com sua função, conforme listado abaixo:

- Mapa: No mapa são apresentados os territórios e os exércitos, sendo estes representados por quadrados coloridos, conforme a cor selecionada pelo jogador. Esses quadrados possibilitam que o jogador possa repor exércitos, selecionar mais de um território ao mesmo tempo para poder realizar ataques e também movimentar exércitos pelo mapa;
- Jogadores: Mostra a lista dos jogadores que fazem parte da partida;
- Ações: As ações são botões que aparecem para o jogador conforme a ação que pode ser realizada no momento. Na Figura 29 o programa está aguardando que um outro jogador termine a sua jogada e passe a vez de jogar ao próximo jogador. Por exemplo, quando o

jogador está no início da jogada, são apresentados dois botões, um com o nome trocar, para trocar a carta, e outro terminar, para informar que a ação de repor exércitos foi terminada;

- Objetivo: Nesse painel é apresentado o objetivo que o jogador deve alcançar para ser o vencedor da partida;
- Próxima Troca de: Neste campo é apresenta a quantidade de exércitos disponíveis para a próxima troca de carta;
- Cartas para a troca: Conforme o andar do jogo, o jogador adquire cartas para poder efetuar sua troca, onde são apresentadas neste painel, e é por onde o jogador pode selecionar as cartas para troca no início de cada jogada; e
- Informações: É o painel que apresenta dados diversos, apresenta informações como quantidade de exércitos a serem colocados nos territórios, valores sorteados no ataque de um território como também a vitória de um dos jogadores da partida.

O cliente sempre faz requisições ao servidor, seja para saber quantos exércitos o jogador tem para colocar nos seus territórios, como para atualizar a sua tela e manter o jogador informado das movimentações de seu oponentes.

4 CONCLUSÃO

O desenvolvimento do projeto teve seu início com a análise das regras do jogo WAR, onde foi possível identificar como o sistema deve atuar, determinando assim suas ações que devem ser executadas de forma automática, como por exemplo, a distribuição de cartas objetivo ao início do jogo e a distribuição dos exércitos no início do jogo, o qual o sistema realiza de forma automática. Outro ponto importante é a forma que os participantes do jogo se conectam ao servidor, que deve ser antes do início da partida. Além da utilização de um ambiente de rede, que é imprescindível para o funcionamento do jogo, por requerer que algumas informações sejam de conhecimento somente do jogador que as possui como, por exemplo, o seu objetivo e as cartas para efetuar uma troca, adquiridas durante o jogo. O que não seria possível caso os jogadores compartilhassem a mesma tela para visualizar a partida, tendo em vista que, essas informações seriam de domínio comum entre os jogadores, e assim, inviabilizando a partida.

Para implementar o projeto, foi escolhida a linguagem Java, levando em conta que, além da utilização do paradigma de programação orientado a objetos, mas principalmente, a facilidade de encontrar informação sobre um determinado tema. Como também a quantidade de fóruns, sites e desenvolvedores que auxiliaram na pesquisa de soluções a problemas como, comunicação, transferência de informações entre sistemas e implementação de agentes. Além disso, a linguagem disponibiliza uma grande variedade de classes para os mais diversos tipos de operações a que podem ser realizadas em meio computacional.

A conexão entre os sistemas cliente e servidor, fora cogitado o uso de duas técnicas. A primeira por troca de mensagens, onde dois sistemas se conectam por intermédio de um endereço IP e *sockets*, efetuando a comunicação pelo intermédio de mensagens textuais. Sendo assim para efetuar a troca de informações entre o sistema cliente e servidor seria necessário padronizar as mensagens, a fim de identificar quais informações estão sendo transmitidas, atualizando objetos do sistema, e ainda, executando operações dentro do sistema e retornando o seu resultado para o sistema que requisitou a informação. Outra forma de conexão entre sistemas, e a escolha para o desenvolvimento do projeto, foi o RMI. A forma de conexão entre os sistemas é a mesma oferecida pela troca de mensagens, por endereço IP e *sockets*. Mas a forma de interação entre os sistemas é feita por intermédio de métodos que são compartilhados pelo servidor, e que são acessados pela aplicação cliente. Com isso em vez de enviar mensagens ao servidor para executar uma determinada

operação, o servidor compartilha a operação e o cliente a executa diretamente. Além disso, é possível transferir, nesta conexão, objetos instanciados tanto do cliente quanto do servidor, e assim não há a necessidade de padronização das informações, tendo em vista que elas já estão no formato dos objetos que são esperados. Além disso, a conexão com base em IP e a utilização de *socket* facilitam o trabalho de comunicação entre os sistemas, pois não necessita outra configuração além do IP, da porta e do nome do serviço que o servidor está utilizando.

Ao analisar como seriam criados os agentes, optou-se pela utilização do JADE que permitiu que o sistema fosse implementado de forma mais rápida e eficaz, já que o JADE possui classes que atendem aos requisitos para a criação de agentes inteligentes, além de ser um *framework* consolidado, assim aumentando a robustez do sistema. Além disso, por implementar padrões internacionais de desenvolvimento de agentes, permite que o sistema entre em um nível de padronização que facilite futuras implementações por parte do autor, ou por terceiros.

Com a utilização do JADE, foram criadas classes que servem para gerenciar os jogadores virtuais, sendo elas:

- Classe Agente: estendendo as funcionalidades da classe Agent do JADE, e contém o comportamento que o jogador virtual deve obedecer;
- Classe ComportamentoPadrão: que estende as funcionalidades do comportamento SimpleBehaviour, e controla as ações que o jogador deve realizar, como também executa as consultas na base de conhecimento do agente.

Então durante a execução do jogo, após os jogadores humanos se conectarem ao servidor, são criados objetos da classe jogador, um para cada jogador virtual, onde são identificados a partir de um atributo, que difere o jogador humano do virtual. Após essa operação, é criado um objeto da classe Agente, onde é incorporando o comportamento do agente, que é um objeto da classe ComportamentoPadrão. Como não é necessário criar um agente para cada jogador virtual, fica a cargo do ComportamentoPadrão reagir conforme o jogador virtual que está jogando. Para isso, ao se determinar qual jogador deve jogar no momento, o gerenciador do servidor verifica, que qual o tipo do jogador, humano ou virtual, e determina para o comportamento do agente a árvore que deve ser utilizada pelo jogador virtual.

A classe ComportamentoPadrão por sua vez não identifica classes de árvores de decisões específicas, somente classes que implementam os métodos da interface Arvore, sendo assim, ao se

criar uma nova classe para representar uma árvore de decisão é necessário incluir na declaração da classe a cláusula “implements Arvore”. Com isso o comportamento do agente fica independente da forma em que a árvore foi implementada, possibilita o uso de qualquer tecnologia, para esse fim, que interage com JAVA.

Outra ferramenta que auxilia na robustez, e na velocidade de implementação do sistema é o JCLIPS. Com ele é possível criar as regras independente do sistema, sendo assim de fácil manutenção, tendo em vista que para acertar alguma regra do agente ou mesmo refinar a árvore de decisão basta alterar o arquivo onde estão escritas as regras sem a necessidade de recompilar o sistema. Dessa forma é possível criar novas versões da inteligência dos jogadores virtuais, como também permite que terceiros criem as suas próprias árvores de decisões. Alguns problemas foram encontrados ao se tentar utilizar essa ferramenta no sistema. As regras são mantidas em arquivo texto, onde as regras devem avaliar alguns valores que o sistema fornece para gerar as decisões. Até o presente momento, não foram encontradas formas de atualizar esses valores conforme o andamento do jogo utilizando a ferramenta. A solução encontrada seria gerar marcações, que devem ser incluídas nas regras, e que representam os valores que deveriam ser avaliados. Um objeto, antes de executar a consulta das regras, leria esse arquivo e substituiria as marcações pelos valores, e assim a consulta seria feita com valores atualizados. Além desse problema, ao se testar o funcionamento do JCLIPS com uma regra de cálculo de média, a ferramenta não reconhece o formato em que foi escrita a regra e ocorre um erro de sintaxe. Ao se tentar o uso da regra de exemplo do JCLIPS, não ocorreu nenhum erro. Foi feito consulta no manual do CLIPS, que apresentou a mesma sintaxe utilizada no exemplo. Para solucionar o problema, foram implementadas as regras na própria linguagem em que foi desenvolvido o sistema.

Com a implementação do jogo em meio computacional, foi possível criar dispositivos para facilitar a interação do jogador com o jogo, disponibilizando apenas as opções que podem ser executadas no momento, além de barrar algumas ações como tentar movimentar exércitos que não fazem fronteira e possibilitar a retirada de uma carta território, apenas se for conquistado um território durante o ataque. Dessa forma facilitando o desenvolvimento da partida, tendo em vista que, as regras já estão implementadas e o jogador não poderá executar ações que não são permitidas em um dado momento, e assim o jogador precisa somente de um conhecimento básico das regras.

Ao se analisar alguns jogos que permitem partidas em rede, foi possível determinar como implementar o jogo com relação aos módulos que deveriam ser desenvolvidos, tendo em vista que

os jogos em rede normalmente possuem um módulo servidor que aceita as conexões de seus clientes. Além disso, foi possível determinar como e quando os jogadores poderiam se conectar ao jogo.

Com a criação do jogo foi possível entender na prática o funcionamento de sistemas distribuídos, como também técnicas para depurar erros para esse tipo de sistema, além de ampliar o entendimento em inteligência artificial, com a pesquisa e implementação de agentes inteligentes e sistemas especialistas. Também foi possível aumentar o conhecimento sobre a utilização de processamento paralelo com o uso de *threads*. E por fim, foi possível identificar defeitos e acertos na gerencia e execução de projetos agregando conhecimento para execução de futuros projetos.

4.1 Trabalhos Futuros

O projeto possui muitos pontos que podem ser alterados para melhorar o seu desempenho como também aperfeiçoamentos para aumentar a sua jogabilidade.

Algumas sugestões para trabalhos futuros são:

Continuar a pesquisa sobre JCLIPS a fim de encontrar a solução do problema citado, implementando as regras do jogo nessa ferramenta.

Fazer pesquisas no campo de estratégia com o intuito de utilizá-las para aperfeiçoar as árvores de decisões dos agentes.

Pesquisar a tecnologia CORBA e o seu uso com RMI possibilitando a implementação do sistema cliente em outras linguagens.

Pesquisar técnicas para otimizar a conexão entre os sistemas, como também desenvolver um método para salvar um jogo possibilitando a continuação de uma partida. Avaliar e implementar formas para aumentar a interação entre os jogadores, como uma forma de envio e recebimento de mensagens através do jogo e mostrar para cada jogador o que está acontecendo no jogo.

Pesquisar novas formas de ataque e defesa, como incluir nível de energia para os exércitos, tipos de exércitos diferentes como infantaria, atiradores de elite e médicos. Onde cada um tem uma especialização, incluindo percentual de acerto no alvo, força e taxa de recuperação.

REFERÊNCIAS BIBLIOGRÁFICAS

AAMODT A. and. PLAZA E. **Case-based Reasoning**: foundational issues, methodological variations and system approaches. AI Communications, 7(1): 39–59 1994.

ARARIBOIA, G. **Inteligência Artificial** – Um curso prático. Rio de Janeiro: Livros Técnicos e Científicos, 1988.

BARR, A. FEIGENBAUM E.A. **The Handbook of Artificial Intelligence**, volume I-II. Willian Kaufmann Inc. Los Altos, Califórnia, 1981.

BAX, M. P. **Uma Proposta de Uso de Agentes e Mapas Conceituais para Representação de Conhecimentos Altamente Contextualizados.** Universidade Federal de Minas Gerais. Departamento de Ciência da Informação. Belo Horizonte, MG: 2001.

BELLIFEMINE F, CAIRE G, TRUCCO T, RIMASSA G. **JADE Programmer's Guide**. 2003 Disponível em http://sharon.cselt.it/projects/jade/doc/programmersguide.pdf

BIGUS, J P. BIGUS J. **Constructing Intelligent Agents Using Java** – Second Edition – Wiley Computer Publishing 2001

BINET, A. **Testes para a Medida do Desenvolvimento da Inteligência**. São Paulo: Melhoramentos, 1929.

BRADSHAW, J. M. **Kaos:** An open agent Architecture Supporting Reuse, Interoperability, and Extensibility, Knowledge Acquisition Workshop (Know'96) Banff, Alberta, Canadá, 1996.

CLARK, A. C. **A Space Odyssey**. Editora Expressão e Cultura, 1968.

CONAI: **Tutorial Inteligência Artificial**. In: CONGRESSO NACIONAL DE AUTOMAÇÃO INDUSTRIAL. Anais: São Paulo, 1994.

CORCHADO, J. *et al*. **Hybrid instance-based system for predicting ocean temperature**. International Journal of Computational Inteligence and Applications, v. 1, n. 1, p. 35-52, 2001.

COSTA, M. T. C. da. **Uma Arquitetura Baseada em Agentes para Suporte ao Ensino a Distância**. Tese (Doutorado em Engenharia da Produção), UFSC – Florianópolis. 1999.

CRIPPA, M. **Sistemas Especialistas**: A Engenharia do Conhecimento Aplicada às Organizações. Trabalho de Conclusão de Curso de Administração da Universidade do Estado de Santa Catarina. – UDESC. 2004

DEITEL, H. M. & P. J. Deitel. **Java: Como Programar** 3a. Edição 2000

DORON, R e PAROT, F. **Dicionário de Psicologia,** Lisboa, Climepsi Editores. (2001)

FARACO, R. A. **Uma Arquitetura de Agentes para negociação dentro do Domínio do Comercio Eletrônico**. Florianópolis, 1998.

FARRENY, H. Les **Systemes Experts: principles et exemples.** Toulouse: CEPADUES-Editions, 1985.

FEIGENBAUM, E. A. **The Handbook of Artificial Intelligence**, Vol. II. Addison-Wesley Publishing, Inc. Massachusetts. (1988)

FERREIRA, A. B. de H. **Dicionário da Língua Portuguesa**. Rio de Janeiro: Editora Nova Fronteira, 1993.

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. 2002. "FIPA Communicative Act Library Specification". FIPA Specification SC00037. http://www.fipa.org/specs/fipa0003. Acesso em Jun 2003.

FLORES, C. D. **Sociedades Artificiais**: a nova fronteira da inteligência nas máquinas. Organizado por Dante Augusto Couto Barone.Capitulo 6 – Fundamentos dos Sistemas Especialistas. Porto Alegre: Bookman, 2003.

FRANCO, C. B. S. de M., YOSHIDA, P. S. **Universidade Java**. São Paulo: Editora Digerati Books, 2004.

GASSER, L. B. **Identity and Aggregation***:* Plurality issuees in multiagent systems. In Eric Werner e Yves Demazeau, editors, Decentralized Artificial Intelligence, páginas 199-212. Elsevier Science, Amsterdam, NL, 2002.

GIESE, L. F. **Estrutura de Agentes para os Processos de Compra e Venda Utilizando Tomada de Decisão Difusa**. Dissertação em Ciências da Computação. Florianópolis, 1998.

GRIMSHAW, D. **Artificial Intelligence Topics with Agents**, 2001. Original Notes of CPS 720. Disponível em: http://www.ryerson.ca/~dgrimsha/courses/cps720/index.html. Acesso em 13 fev. 2003.

GONÇALVES, C. A. **Aquisição e Representação do Conhecimento para Sistemas Especialistas**. Faculdade de Economia e Administração da Universidade de São Paulo, SP, 1986.

GROW. **Histórico do Jogo.** 1997. Disponível: URL: http://www.grow.com.br/historia/centro.htm acessado em 03/09/2005.

HARMON, P. KING, D. **Sistemas Especialistas**. A inteligência artificial chega ao mercado. Rio de Janeiro: Campus, 1988.

HELIN, H. LAUKKANEN, M. **Performance Analysis of software agent comunication in slow Wirelerss**, IEEE, International Conference on Computer Communications and Networks, 2002.

IBM. **Antivirus online**: scientific papers. 1998. Disponível: site IBM anti-virus online. URL: http://www.av.ibm.com/InsideTheLab/Bookshelf/ScientificPapers/index.html. Consultado em: 19 ago. 1999.

JADE, Cselt http://sharon.cselt.it/projects/jade Parma - Itália. 2003.

JENNINGS, N. R., WOOLDRIDGE, M., KINNY, D. **A Methodology for Agent-Oriented Analysis and Design.** Proc. 3rd Int Conference on Autonomous Agents (Agents-99) Seattle,WA. 28, 1998.

KERN, E. **Uma Estrutura de Agentes para o Processo de Licitação**. Dissertação (Mestrado em Ciência da Computação) - UFSC, 1998.

KOLODNER J. **Case-Based Reasoning**. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

KURZWEIL, R. **The age of Intelligent Machines**. MIT Press, Cambridge, Massachusetts. 1990.

LEMOS, A. GIRARDI, R. FERREIRA, S. ABARFI **Uma Arquitetura Reutilizável para a Recuperação e Filtragem de Informação**, 2002.

LINDEMANN, V. **Sociedades Artificiais**: a nova fronteira da inteligência nas máquinas. Organizado por Dante Augusto Couto Barone. Porto Alegre: Bookman, 2003.

LINO T. A. L. R. **Inteligência Artificial, Humana e a Emoção –** Universidade de Lisboa-Portugal , 2004.

LOH S. & GARIN R S. **Web Intelligence –** Inteligência Artificial Para Descoberta de Conhecimento Na Web**.** 2004.

LUGER, G.F; STUBBLEFIELD, W.A. **Artificial Intelligence**: Structures and Strategies for Complex Problem Solving. Benjamim/Cummings, Redwood City, Califórnia, second edition. 1993.

LUSTOSA V. G. **O Estado da Arte em Inteligência Artificial.** Colabor@ - Revista Digital da CVA-Ricesu **ISSN** 1519-8529, vol.2 – nº8: Brasília (D.F), 2004.

LUCENA, P. **Semantic Agent, uma plataforma para desenvolvimento de agentes inteligentes**. USP - São Carlos, 2003.

MAES, P. **Artificial Life Meets entertainment**: lifelike autonomous agents communications of ACM, 1995.

MCCARTHY, J. **What is Artificial Intelligence**? Disponível em: <http: //www-formal.stanford.edu/jmc/whatisai/whatisai.html> . Acesso em: out. 2002

MARTIN, D. L. CHEYER, A. J. and MORAN, D. B. **The open agent architecture**: a framework for building distributed software systems, Applied Artificial Intelligence, 1999.

MATURANA, H. R. **Da Biologia à Psicologia** Artmed Editora LTDA, 1998.

MICHAELIS - **Dicionário da Lingua Portuguesa.** Companhia Melhoramentos, 1998.

MINSKY, M A. **Framework for Representing Knowledge**. In: Winston, Patrick H. Ed. The Psychology of Computer Vision. McGraw-Hill, New York. 1975.

NEGROPONTE, N. **Beeing Digital**. Ed. Vintage, 1995.

NEWELL, A; SIMON, H. **Computer Science as Empirical Enquiry**: Symbols and Search. In Haugeland, John (ed.) MIT Press, Massachusetts: 1976

NISSEN, **Mark. Intelligent Agents**: A Technology and Business Application Analysis. Disponível na Internet via WWW. URL: http://haas.berkeley.edu/~heilmann/agents/1995.

PEREIRA S. do L. **Inteligência Artificial.** Artigo Cientifico**:** Área de Inteligência Artificial – USP 2003.

PIAGET, J. **O Nascimento da Inteligência na Criança**. 4. ed. Rio de Janeiro: Zahar, 1982.

QUIBEL, J.; ROGER.M. **Les Systèmes Experts dans 'Entreprise: pourquoi, comment?.** Paris: Les Éditions d'Organisation, 1989.

REINGOLD, E; NIGHTINGALE, E. **Works in AI**, 1999.

REZENDE, S.O. **Sistemas Inteligentes**: Fundamentos e Aplicações. Barueri, SP: Manole, 2003.

RICH, E; KNIGHT, K. **Inteligência artificial**. São Paulo : Markron Books do Brasil, 1994

RILEY, G. **CLIPS**: a tool for building expert systems. Home P a g e. 2 0 0 5. Disponível em http://www.ghg.net/clips/CLIPS.html Acessado em Abr. 2007.

ROCHA, A. de R. **Opiniões, artigos e publicações**. Disponível em: http://www.comp.ufla.br/~undersun. Acesso em: 18 de Nov. de 2002.

ROOK, F.W., GROGHAN, J.W. **The Knowledge Acquisition Activity Matrix** : a System Engeneering Conceptual Framework. Systems, Man and Cybernetics, Vol 19. 1989.

RUSSEL, S. NORVIG, P. **Artificial Intelligence**. A Modern Aproach, Ed. Prentice Hall, 1995.

SCHALKOFF, R.J. **Artificial Intelligence***:* An Engineering Approach. McGraw-Hill, New York. 1990.

SELLITTO, M. A**. Artificial Intelligence**: an application in a continuous process industry Gest. Prod. vol.9 no.3 São Carlos Dec. 2002

SELKER, T. **Afetando o Humanity**. Vol. 44. Imprensa York: EUA, 2001.

SILVA, L.A. M. **Estudo e Desenvolvimento de Sistemas Multiagentes usando JADE**: Java Agent Development framework.Trabalho de Conclusão de Curso UNIFOR. 2003.

SOUZA, E. M. S. de. **Uma Estrutura de Agentes para Asseoria na Internet**. Dissertação (Mestrado em Engenharia de Produção) – Centro Tecnologico, UFSC, Florianópolis, 1996.

TURING, A. **The imitation game, Computing Machinery and Intelligence Mind**. Vol 59, nº 236, 1950.

WOOLDRIDGE, M. **Intelligent Agents**. In Weiss, G. ed., Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence. The MIT Press, 1999.

WOOLDRIDGE, M. JENNINGS, N. **Agent Technology** – Foundations, Applications, and Markets. Springer-Verlag, Berlin, Heidelberg, New York. (1998)

WANGENHEIM, C.G.Von. **Case-Based Reasoning** – A Short Introduction. Relatório Técnico, Universidade do Vale do Itajaí - UNIVALI, 2002.

5 GLOSÁRIO

Multiplayer	Termo usado em jogos para designar um modo em que jogadores participam de um jogo em uma mesma partida
Repor	Momento em que o jogador adquiri mais exércitos e os distribui no mapa.
Rodada	Parte do jogo em que o jogador repõe exércitos no mapa, ataca e movimenta o seu exército.
Singleplayer	Termo usando em jogos para designar um modo em que um jogador desenvolve a história do jogo sozinho.
Socket	Portas virtuais para conexão de programas em um computador via um protocolo de rede.

APÊNDICES

Diagramas

Segue diagramas do sistema que demonstram como o sistema deve funcionar.

A.1 DIAGRAMAS DE ATIVIDADES

Este diagrama mostra as ações que o usuário pode realizar no sistema e como ele deve responder.

A.1.1 Inicializa Jogo (Servidor)

O servidor serve para gerenciar a partida, o diagrama abaixo mostra como será o funcionamento do sistema ao ser criado um novo jogo. O usuário entra com dados como nome e cor de exército a ser utilizado na partida, e defini quantos jogadores farão parte dela. Em seguida deve esperar os participantes se conectarem de mostrarem que estão prontos, para que o servidor possa dar início a partida.

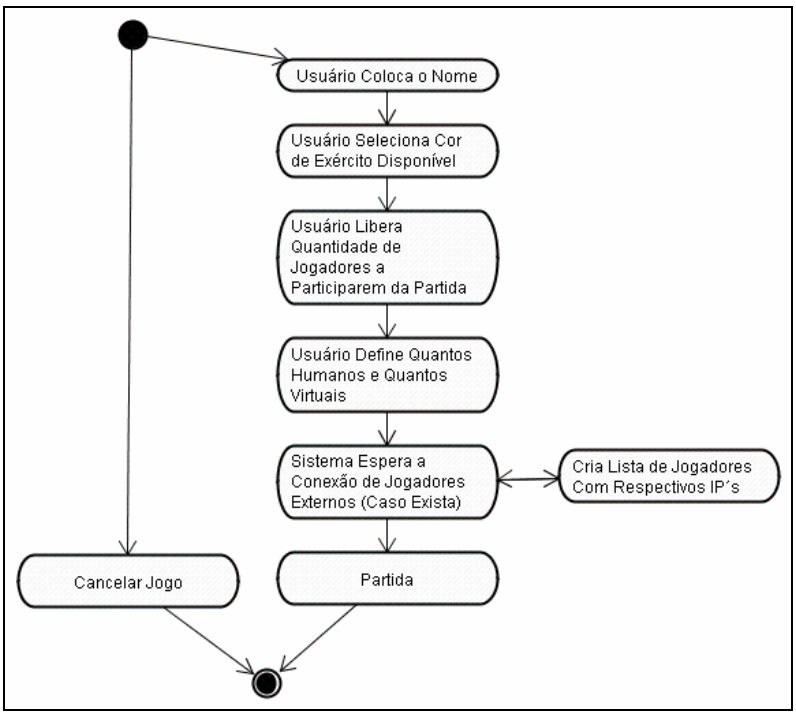
Usuário Coloca o Nome
Usuário Seleciona Cor de Exército Disponível
Usuário Libera Quantidade de Jogadores a Participarem da Partida
Usuário Define Quantos Humanos e Quantos Virtuais
Sistema Espera a Conexão de Jogadores Externos (Caso Exista)
Cria Lista de Jogadores Com Respectivos IP´s
Cancelar Jogo
Partida

Figura 30. Diagrama de Atividade - Inicializa Jogo (Servidor)

A.1.2 Inicializa Jogo (Cliente)

O cliente é todo aquele que fizer parte de uma partida e que não seja o servidor. O diagrama abaixo mostra como deverá ser portar o sistema ao fazer parte de uma partida criada por um servidor. O usuário entra com o IP e porta do servidor, onde o cliente fará a conexão com o servidor a fim de verificar se o servidor está iniciado, em seguida entra com os dados de nome e cor de exército, após indica se está pronto para iniciar uma partida, e fica aguardando o servidor iniciá-la.

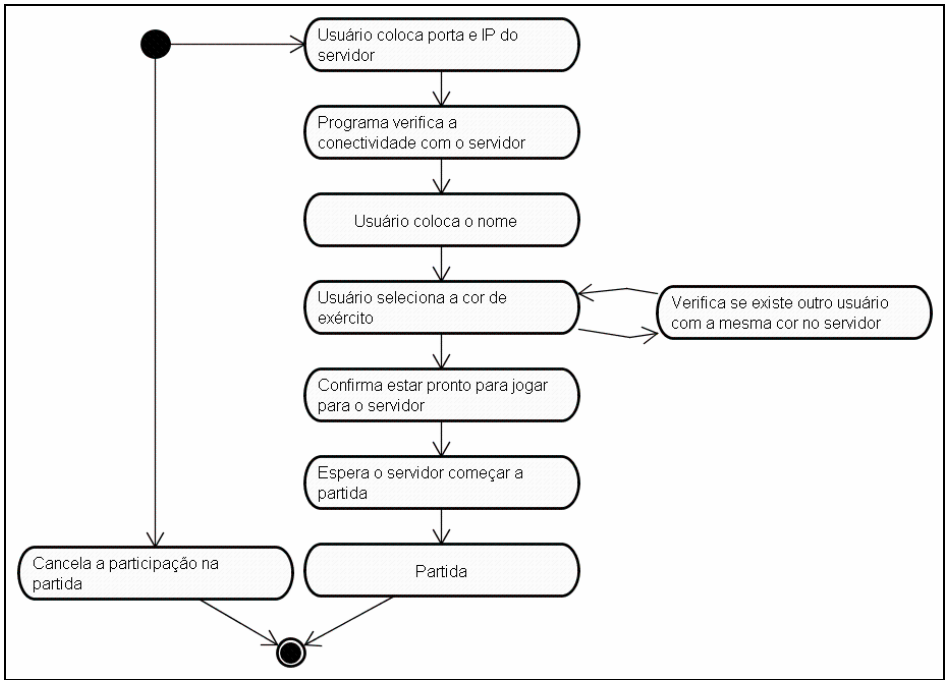
Usuário coloca porta e IP do servidor
Programa verifica a conectividade com o servidor
Usuário coloca o nome
Usuário seleciona a cor de exército
Verifica se existe outro usuário com a mesma cor no servidor
Confirma estar pronto para jogar para o servidor
Espera o servidor começar a partida
Cancela a participação na partida
Partida

Figura 31. Diagrama de Atividade - Inicializa Jogo (Cliente)

A.1.3 Partida

O decorrer de uma partida de jogo segue algumas regras, ao qual o sistema deve monitorar para que sejam respeitadas. Diagrama representado pela Figura 32, mostra como funcionará o decorrer de uma partida. O sistema inicia distribuindo as cartas território para os participantes, e logo em seguida distribui os objetivos para os participantes. A seguir o sistema distribui um exército de cada participante nos seus devidos territórios. O sistema determina quem deve começar a jogar e permite a cada jogador, um por vez, repor exércitos nos seus devidos territórios. O sistema fará a verificação das cartas que o jogador tem para troca, e dessa forma liberando uma opção para

ele trocar ou não as cartas, sendo que quando a quantidade de cartas for de 5, ele forçará a troca das mesmas. Em seguida o jogador terá a opção de parar a sua jogada ou atacar um território inimigo. Se optar por atacar, ele deverá escolher um território origem, de sua posse, e um território destino, de posse de outro exército, para o ataque. Após cada ataque que o jogador fizer, terá a opção de parar ou continuar, sendo que ficará impossibilitado de atacar caso conquiste o território ou fique sem exércitos para ataque. Caso o jogador opte por parar sua rodada de ataques, ele poderá movimentar os exércitos pelos territórios de sua posse e eventualmente pegar uma carta para troca. Após essa ação será permitido ao próximo jogador realizar todo o processo, e assim por diante até que um dos jogadores alcance o seu objetivo.

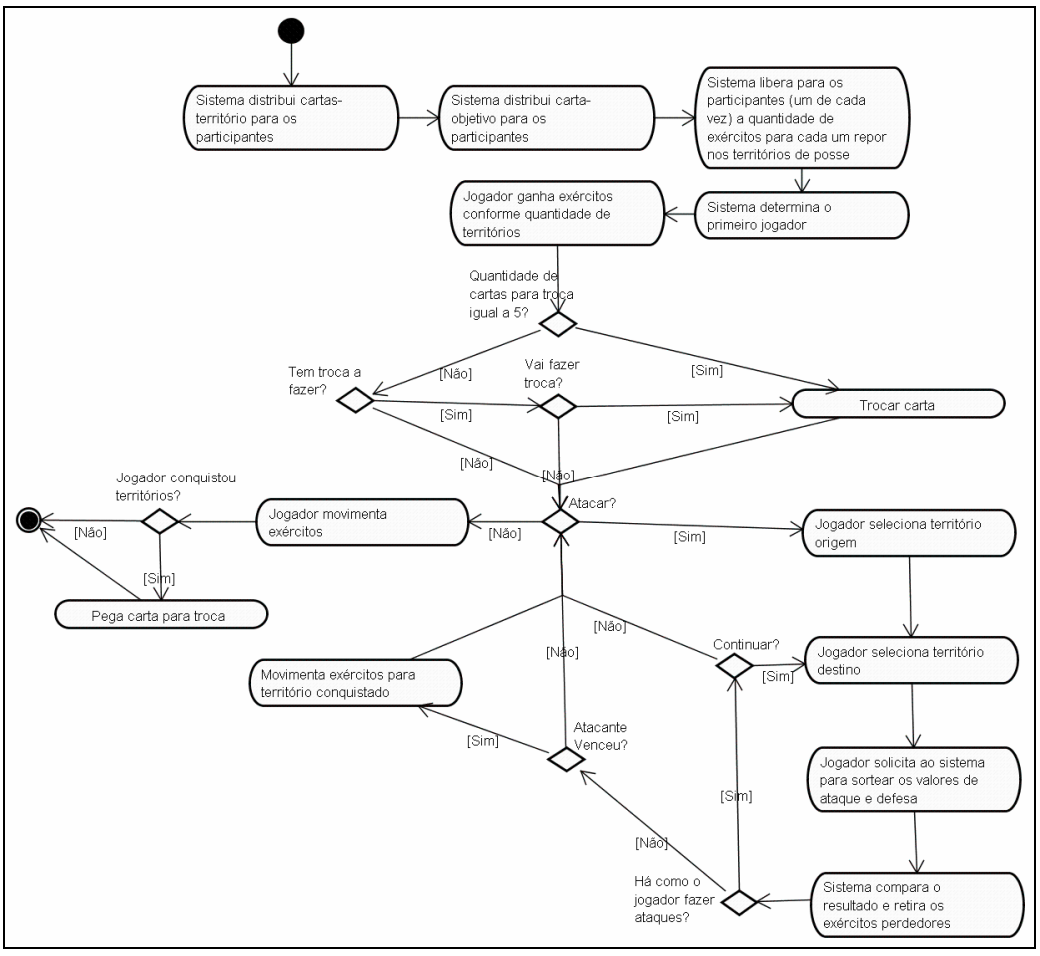

Figura 32. Diagrama de Atividade – Partida

A.2 DIAGRAMAS DE ESTADO

Os diagramas de estado mostram o comportamento do sistema conforme o decorrer do tempo. A seguir serão mostrados diagramas que mostram o comportamento do sistema ao iniciar uma partida, durante o jogo e a comunicação entre computadores.

A.2.1 Iniciar Jogo

O sistema oferece dois tipos de inicio de jogo, como servidor e como cliente. Quando o sistema é iniciado como servidor, é necessário que fique aguardando até que os demais jogadores se conectem e sinalizem que estão prontos para iniciar a partida. Em contra partida o cliente deve conectar com o servidor e logo estando pronto, deve mandar uma mensagem ao servidor, e dessa forma o servidor inicia a partida.

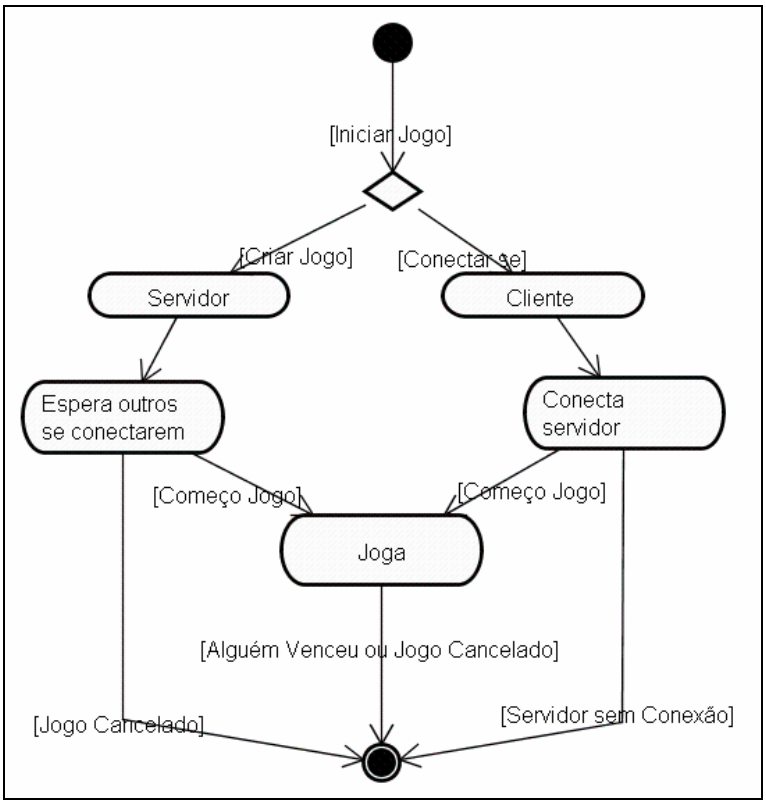

Figura 33. Diagrama de Estado - Iniciar Jogo

A.2.2 Durante o Jogo

Ao início de cada rodada, onde um jogador deve repor exércitos, atacar, movimentá-los e depois parar, o sistema deverá reconhecer a quantidade de exércitos que são de direito do jogador, a quem o mesmo vai atacar, isso se o jogador desejar atacar, se ganhou territórios (dando o direito de o jogador receber uma carta para troca), se está parando, possibilitando assim movimentar exércitos entre os territórios que fazem divisa e entregar uma carta para troca a ele.

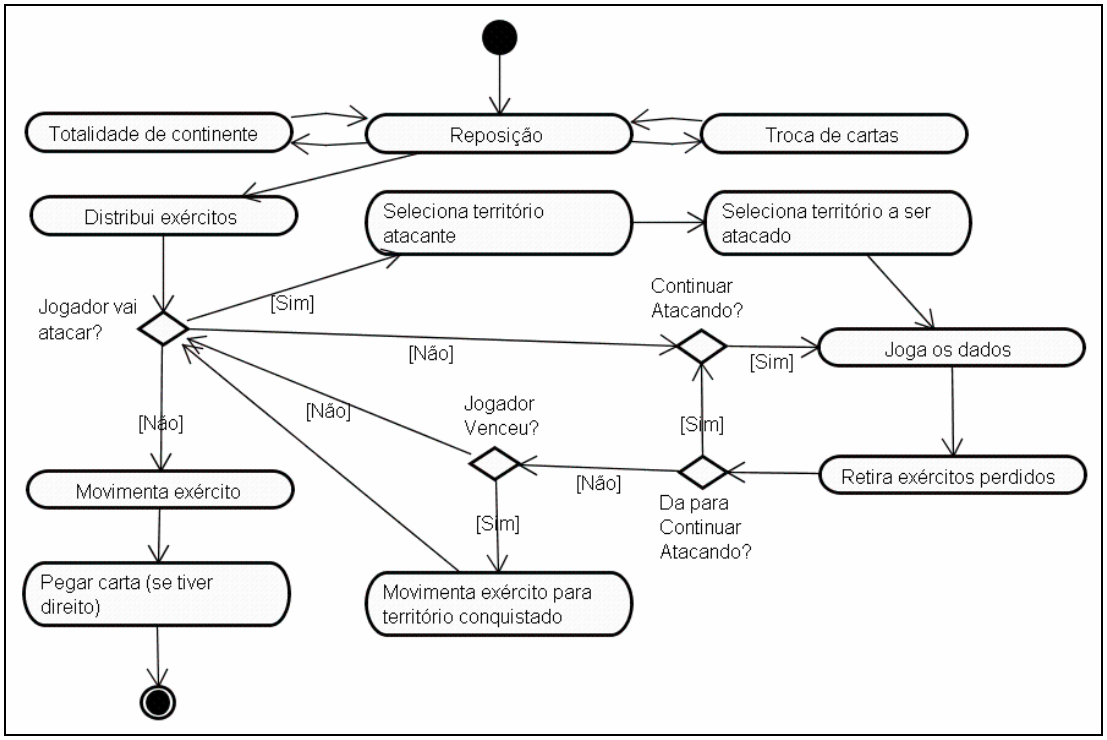

Figura 34. Diagrama de Estado - Durante o Jogo

A.2.3 Comunicação entre Computadores

A comunicação entre o cliente e servidor se dará por intermédio de mensagens utilizando *socket*. Então o cliente envia uma mensagem ao servidor, que por sua vez processa essa mensagem e retorna ao cliente. Esse ciclo continua até que o cliente feche a conexão.

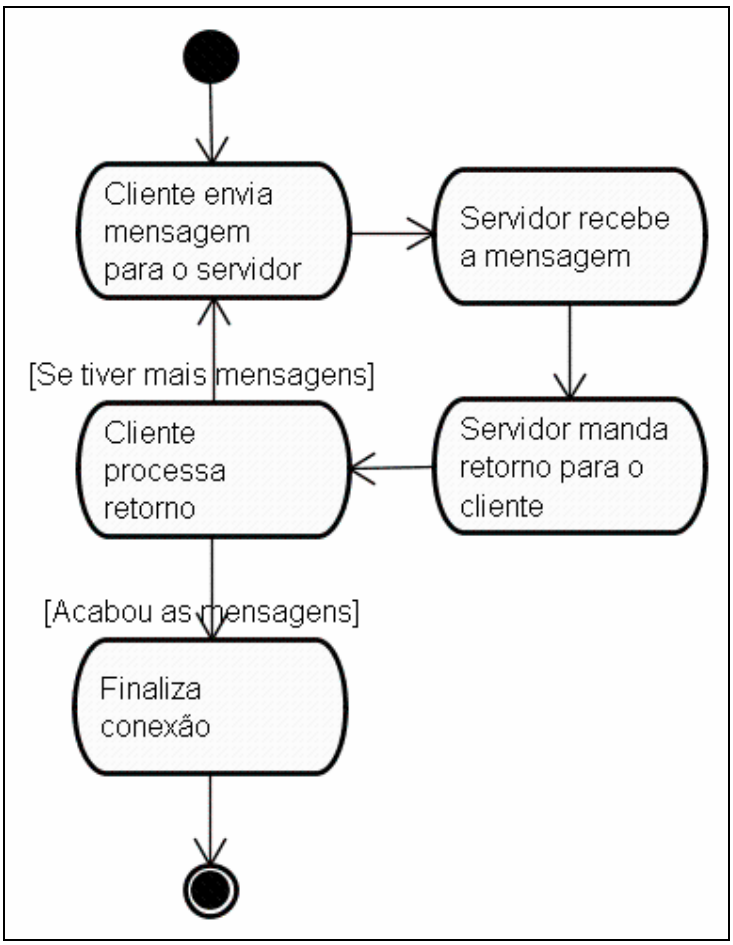
Cliente envia mensagem para o servidor
Servidor recebe a mensagem
[Se tiver mais mensagens]
Cliente processa retorno
Servidor manda retorno para o cliente
[Acabou as mensagens]
Finaliza conexão

Figura 35. Diagrama de Estado - Comunicação Entre Computadores

A.3 DIAGRAMA DE SEQÜÊNCIA

O jogador informa o endereço do servidor para o cliente, logo o servidor deve responder informando que está pronto para receber e enviar mensagens. Em seguida o cliente e o servidor ficam trocando mensagens e assim se atualizando. Após todas as mensagens forem trocadas, o cliente finaliza a conexão.

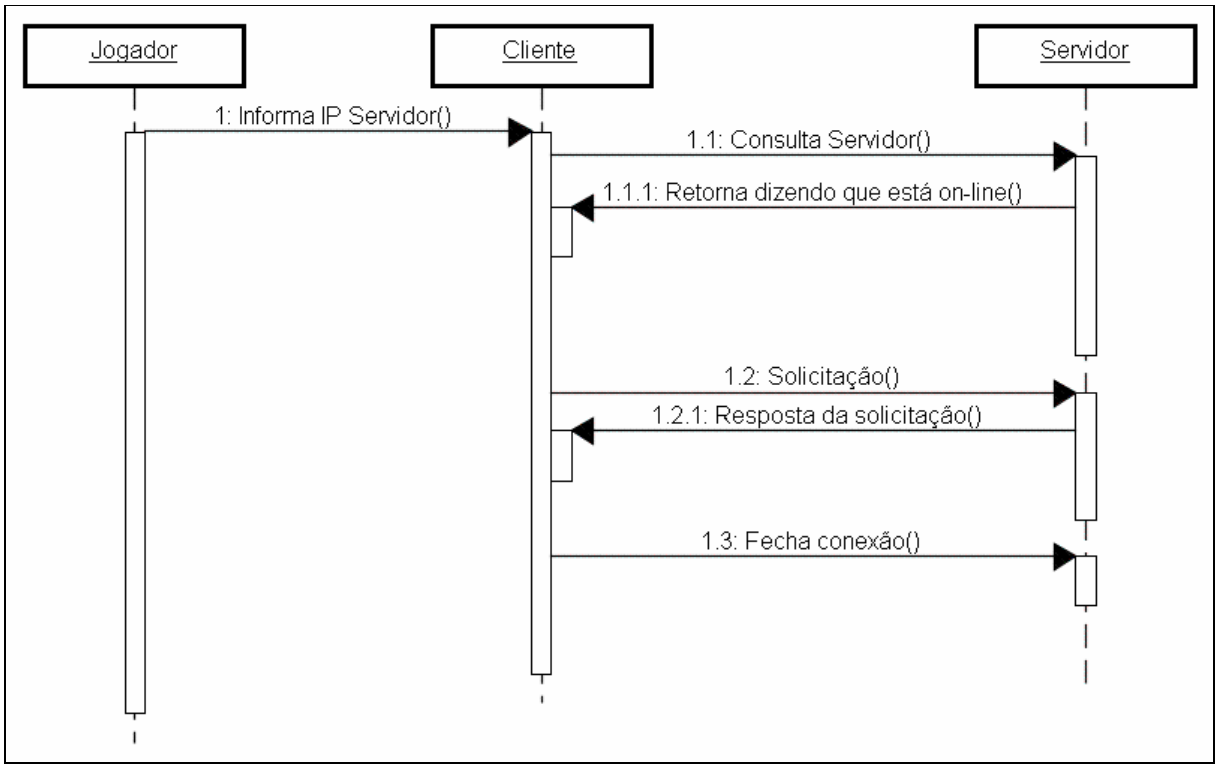

Figura 36. Diagrama de Seqüência – Comunicação Cliente-Servidor

A.4 ÁRVORES DE DECISÃO

A.4.1 Árvore Genérica

A árvore genérica serve para ser utilizada pelos objetivos que não possuem uma árvore de decisão própria, a Figura 37 ilustra isso.

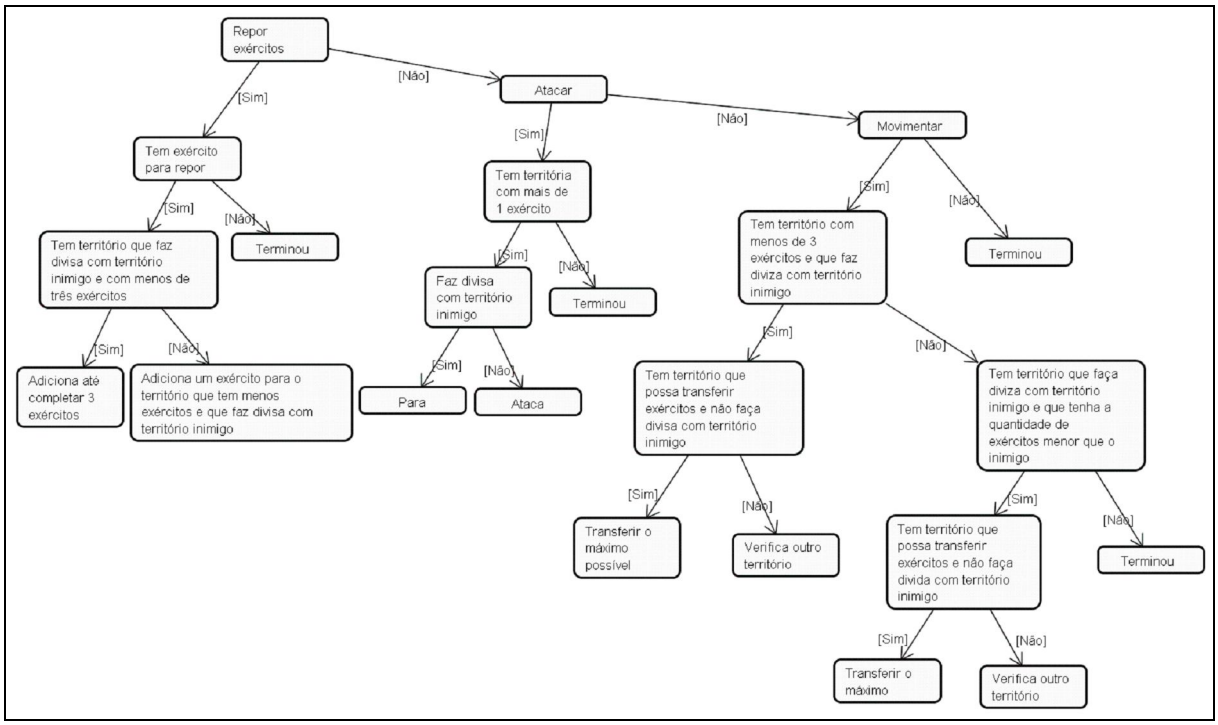

Figura 37. Árvore de Decisão – Árvore Genérica